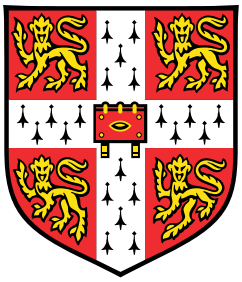
Advances in Probabilistic Modelling: Sparse Gaussian Processes, Autoencoders, and Few-shot Learning



Matthias Bauer

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Pembroke College

September 2019

DECLARATION

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee

ABSTRACT

Learning is the ability to generalise beyond training examples; but because many generalisations are consistent with a given set of observations, all machine learning methods rely on inductive biases to select certain generalisations over others. This thesis explores how the model structure and priors affect the inductive biases of probabilistic models, and our ability to learn and make inferences from data.

Specifically we present theoretical analyses alongside algorithmic and modelling advances in three areas of probabilistic machine learning: sparse Gaussian process approximations and invariant covariance functions, learning flexible priors for variational autoencoders, and probabilistic approaches for few-shot learning. As inference is rarely tractable, we discuss variational inference methods as a secondary theme.

First, we disentangle the theoretical properties and optimisation behaviour of two widely used sparse Gaussian process approximations. We conclude that a variational free energy approximation is more principled and extensible and should be used in practice despite potential optimisation difficulties. We then discuss how general symmetries and invariances can be integrated into Gaussian process priors and can be learned using the marginal likelihood. To make inference tractable, we develop a variational inference scheme that uses unbiased estimates of intractable covariance functions.

We then address the mismatch between aggregate posteriors and priors in variational autoencoders and propose a mechanism to define flexible distributions using a form of rejection sampling. We use this approach to define a more flexible prior distribution on the latent space of a variational autoencoder, which generalises to unseen test data and reduces the number of low quality samples from the model in a practical way.

Finally, we propose two probabilistic approaches to few-shot learning that achieve state of the art results on benchmarks, building on multi-task probabilistic models with adaptive classifier heads. Our first approach combines a pre-trained deep feature extractor with a simple probabilistic model for the head, and can be linked to automatically regularised softmax regression. The second employs an amortised head model; it can be viewed to meta-learn probabilistic inference for prediction, and can be generalised to other contexts such as few-shot regression.

The highest forms of understanding we can achieve
are laughter and human compassion.
— Richard P. Feynman

ACKNOWLEDGEMENTS

I am grateful to my supervisors Carl Rasmussen and Bernhard Schölkopf, who accepted me into their labs without much prior experience in machine learning. Their expertise, guidance, and encouragement have helped me to grow both as a researcher in this area as well as a person. I am equally grateful to Richard Turner, with whom I worked closely during the second half of my studies, and whose inspiration, insights, and advice have guided me throughout my PhD and beyond.

I am very fortunate to have met and worked with many inspiring people both at the Machine Learning Group in Cambridge as well as at the Empirical Inference Department in Tübingen. I am thankful to every member of the labs for the ambitious yet kind and up-lifting atmosphere. I would like to thank all of my co-authors for the fruitful collaborations, in particular Mark van der Wilk, whose fresh perspective on probabilistic modelling has helped to shape my own, as well as Jonathan Gordon, John Bronskill, Andriy Mnih, and Mateo Rojas-Carulla.

My PhD was funded by the Max Planck Society, a Qualcomm Studentship in Technology, and the UK Engineering and Physics Research Council (EPSRC). Throughout my undergraduate and graduate studies I was supported by scholarships from the Maximilianeum Foundation, the German Academic Scholarship Foundation (Studienstiftung des Deutschen Volkes), the Max Weber-Program of the state of Bavaria, and the German Academic Exchange Service (DAAD). In particular I thank Hanspeter Beisser for his guidance and support.

This work would not have been possible without a number of mostly Open Source projects and tools such as GNU/Linux, Python, Tensorflow, GPflow, LaTeX, or Ti*k*Z. We should never take them for granted.

Finally, I thank my family and friends for their support and kindness.

Matthias Bauer
Tübingen, September 2019

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

0 INTRODUCTION

Probabilistic machine learning takes into account uncertainty on a fundamental level; its probabilistic framework explains how to represent, reason about, and manipulate uncertainties. Probabilistic modelling has therefore become a cornerstone of "scientific data analysis, machine learning, robotics, cognitive science, and artificial intelligence" (Ghahramani, 2015); fields in which the understanding and consideration of uncertainties is paramount.

In the probabilistic framework we describe our beliefs about the world and our models using probability theory. By writing down a probabilistic model, we express the connections between the known knowns, i.e. observations, and known unknowns, such as latent structures, latent variables, or random noise. Using Bayes' rule we can infer likely values and even full distributions of unknown quantities from known quantities.

When facing a particular problem or task, the two main questions in probabilistic modelling and machine learning are (i) how to choose the probabilistic model describing the data, and (ii) how to perform probabilistic exact or approximate inference in this model. While model specification and inference are separate steps, they are often linked, as the complexity of the inference step depends on the model structure and distributions.

In general, we specify a probabilistic model by specifying a joint distribution of observations (known quantities) and latent variables (unknown quantities). Equivalently, we can define a prior distribution on the latent variables and a probability distribution that links them to the observations, which is termed the likelihood of the latent variables.

The choice of model, its structure, and particularly its prior heavily influence the encoded implicit or explicit inductive biases and, therefore, our ability to learn and generalise from data as well as the data efficiency associated with it. In principle, a large number of possible generalisations are consistent with a set of observations such that we need to make certain assumptions as specified through inductive biases to reduce them to a small and plausible set (Mitchell, 1980). Consider as an example the universal approximation theorems that famously state, that a single- or multi-layer feed-forward neural network of sufficient but finite size can approximate any continuous function with compact support (Cybenko, 1989; Hornik et al., 1989). However, for an arbitrary function it is unclear how to learn the corresponding weights, that is, which of the many model configurations to prefer. Yet by choosing a model with the right inductive biases for the problem

at hand, we can accurately learn relatively complex functions from small amounts of data. In other words, the model structure and its priors make it more prone to model certain functions than others. The choice and number of hidden layers and units, their weight sharing (such as convolutional layers), and non-linearities influence the typical functions represented by a neural network. Similarly the choice of covariance function affects all properties of typical sample draws from a Gaussian process, such as smoothness or stationarity.

In this thesis we present theoretical analyses as well as algorithmic and modelling advances in three areas of probabilistic machine learning. They are all connected by the underlying question of how structural choices and priors affect the inductive biases of models, and our ability to learn and make inferences from data for the tasks at hand.

These choices also determine the tractability of probabilistic inference and the approximations that are likely to be successful. As a secondary theme, we therefore discuss approximate probabilistic inference through variational methods and how it affects learning in these cases.

The three areas of probabilistic machine learning addressed in this thesis are (i) supervised learning with sparse Gaussian processes and how to incorporate general invariances into their prior covariance functions; (ii) unsupervised learning with variational autoencoders and the role of their prior; and (iii) probabilistic models for supervised few-shot learning tasks.

OVERVIEW AND MAIN CONTRIBUTIONS

This section gives a brief overview of these three topics. Within each topic we highlight our main results and present how they relate to the two main themes of this thesis: inductive biases through structure and priors, and approximate variational inference.

These results have led to articles which I co-authored with my supervisors and other colleagues in the community, and have been published at international conferences and workshops. Below, I cite the main references for each chapter; my own contributions are highlighted at the beginning of the respective chapters in the main part of the thesis.

1. Sparse Gaussian processes and invariant covariance functions

Gaussian process (GP) models are an important class of Bayesian non-parametric models. Originally developed for supervised regression tasks, they directly model the underlying latent regression function. The properties of these functions such

as smoothness, stationarity, or scales of typical fluctuation are defined by the kernel or covariance function of the GP. Due to these strong inductive biases, learning with GPs can be very data efficient and provide principled uncertainties. While inference in GP regression with Gaussian likelihoods is tractable, it is intractable for general likelihoods such as the Bernoulli or Multinomial used in GP classification. Moreover, even fully tractable inference may be computationally infeasible as the necessary inversion of the covariance matrix scales cubically with the size of the data. Therefore, two important research questions are: (i) what are the right inductive biases/how to choose the covariance functions and its properties; and (ii) how to make probabilistic inference in GP models tractable and scalable to large datasets.

- In Chapter 3 we disentangle the theoretical properties and optimisation behaviour of two widely used sparse Gaussian process approximations, which are both based on a low-rank approximation of the exact covariance matrix. Our results allow us to explain earlier empirical findings in more detail. Moreover, we conclude that one of the approaches, a variational free energy approximation to the full Gaussian process model, is more principled and extensible and should be used in practice. This is joint work with Mark van der Wilk and Carl E. Rasmussen and was originally published as 'Understanding Probabilistic Sparse Gaussian Process Approximations' (Bauer et al., 2016).

- In Chapter 4 we focus on the Gaussian process priors and discuss how general symmetries can be integrated into GP priors *and* can be learned using the marginal likelihood. To make learning tractable in this setting, we develop an approximate inference scheme that makes use of the variational approach discussed earlier alongside several other recent improvements to GP inference for regression and classification. Our main finding is that the marginal likelihood can indeed be used to identify and learn symmetries and invariances from data. These results are joint work with Mark van der Wilk, ST John, and James Hensman and were originally published as 'Learning Invariances using the Marginal Likelihood' (van der Wilk et al., 2018).

2. Flexible priors for variational autoencoders

A prominent class of unsupervised learning models are variational autoencoders (VAEs). Despite their name and superficial similarities to classical autoencoders, they are actually deep generative latent variable models. While probabilistic inference through amortisation is scalable, we can still ask how to incorporate structure, what the right inductive biases are, and how they should affect the

design of the model. A VAE model is specified by fixing a prior distribution on latent codes as well as a likelihood of the latent codes for the corresponding observations.

- Prompted by recent research we discuss the role of the prior in variational autoencoders (VAEs) in Chapter 5 and propose a mechanism to define flexible distributions using a form of rejection sampling that we refer to as *learned accept/reject sampling* (Lars). We then use this approach to define a more flexible prior distribution on the latent space of a VAE, which generalises to unseen test data and reduces the number of low quality samples from the model. These findings are joint work with Andriy Mnih and were first published as 'Resampled Priors for Variational Autoencoders' (Bauer and Mnih, 2019).

3. Probabilistic approaches to few-shot learning

In the last part of the thesis, we propose two probabilistic approaches to few-shot learning. The aim of a few-shot learning task is to classify instances based on very few examples per class with access to a large repository of related tasks on different classes. Because of the limited number of examples, uncertainty is rife, such that a probabilistic approach is appropriate. The two predominant perspectives on this task are: (i) transfer learning and (ii) meta-learning. Transfer learning addresses how previously acquired knowledge and representations can be leveraged to improve performance on a new but related task. Meta-learning instead emphasises meta-algorithms that can adapt a model or learner to new tasks, treating the training repository as a collection of many smaller tasks.

- In Chapter 7 we follow the transfer learning perspective and show that a surprisingly simple probabilistic model for the head of a pretrained deep classifier often works well for this task and can outperform more complicated approaches. We can recast a special case of our model as automatically regularised softmax regression; the automatic regularisation also allows for efficient and balanced online learning when evaluating not only on new classes but also on the base classes. The probabilistic model for concept transfer is key and inductive biases in the model as well as the choice of the prior influence the final performance. This chapter is based on joint work with my co-first author Mateo Rojas-Carulla as well as Jakub Świątkowski, Bernhard Schölkopf, and Richard Turner and was published as 'Discriminative k-shot learning using probabilistic models' (Bauer*, Rojas-Carulla* et al., 2017b).

- While our previous approach requires a pretrained model as well as meta-test time optimisation, in Chapter 8 we present an extended model in

the vein of meta-learning. It is trained episodically and allows for fast meta-test time adaptation through amortised inference of the head model. Through episodic training and the model structure, we *implicitly* learn a prior for the head weights, resulting in state-of-the-art performance on several benchmark datasets. These results are joint work with Johnathan Gordon, John Bronskill, Sebastian Nowozin, and Richard Turner and were published as 'Meta-Learning Probabilistic Inference for Prediction' (Gordon*, Bronskill* et al., 2019).

AUTOMATIC ESTIMATION OF MODULATION TRANSFER FUNCTIONS

In addition to Bayesian probabilistic inference, I have also carried out research in the field of computational photography. This work is not covered in this thesis. Most notably, we have developed a method to estimate a physical quality metric of a camera and its lens, the so-called *modulation transfer function*, directly from a set of photographs rather than measuring it in an optics laboratory. This work and related projects involved building custom hardware to collect a data set of ground truth lens properties, capturing a data set of photographs for training and evaluation, as well as designing and training the entire machine learning pipeline. These results are joint work with Michael Hirsch, Valentin Volchkov, and Bernhard Schölkopf and were published as 'Automatic estimation of modulation transfer functions' (Bauer et al., 2018).

LIST OF PUBLICATIONS

The following provides a list of all publications I have co-authored throughout my PhD, regardless of whether they appear in this thesis or not.

Conference proceedings:

Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen (2016). 'Understanding Probabilistic Sparse Gaussian Process Approximations'. In: *Advances in Neural Information Processing Systems 29*

Matthias Bauer, Valentin Volchkov, Michael Hirsch, and Bernhard Schölkopf (2018). 'Automatic estimation of modulation transfer functions'. In: *2018 IEEE International Conference on Computational Photography (ICCP)*

Mark van der Wilk, Matthias Bauer, ST John, and James Hensman (2018). 'Learning Invariances using the Marginal Likelihood'. In: *Advances in Neural Information Processing Systems 31*

Jonathan Gordon*, John Bronskill*, Matthias Bauer, Sebastian Nowozin, and Richard Turner (2019). 'Meta-Learning Probabilistic Inference for Prediction'. In: *Proceedings of the 7th International Conference on Learning Representations*. *equal contribution

Matthias Bauer and Andriy Mnih (2019). 'Resampled Priors for Variational Autoencoders'. In: *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*

Frederik Harder, Matthias Bauer, and Mijung Park (2020). 'Interpretable and Differentially Private Predictions'. In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*. New York, USA: AAAI Press

Journal articles:

Matthias Bauer*, Johannes Knebel*, Matthias Lechner, Peter Pickl, and Erwin Frey (2017a). 'Ecological feedback in quorum-sensing microbial populations can induce heterogeneous production of autoinducers'. *eLife*. *equal contribution

Workshop contributions:

Matthias Bauer*, Mateo Rojas-Carulla*, Jakub B. Świątkowski, Bernhard Schölkopf, and Richard E. Turner (2017b). 'Discriminative k-shot learning using probabilistic models'. In: *Second Workshop on Bayesian Deep Learning at the 31st Conference on Neural Information Processing Systems*. *equal contribution

Jonathan Gordon*, John Bronskill*, Matthias Bauer*, Sebastian Nowozin, and Richard E. Turner (2018b). 'Versa: Versatile and Efficient Few-shot Learning'. In: *Third Workshop on Bayesian Deep Learning at the 32nd Conference on Neural Information Processing Systems*. *equal contribution

Jonathan Gordon*, John Bronskill*, Matthias Bauer*, Sebastian Nowozin, and Richard E. Turner (Dec. 2018a). 'Consolidating the Meta-Learning Zoo: A Unifying Perspective as Posterior Predictive Inference'. In: *Workshop on Meta-Learning (MetaLearn 2018) at the 32nd Conference on Neural Information Processing Systems*. *equal contribution

1

BACKGROUND

In this chapter, we provide the necessary background for this thesis. We give a high-level introduction to probabilistic machine learning (Section 1.1) and highlight the role of the marginal likelihood for model selection and training (Section 1.2). We then discuss variational inference (Section 1.3) before introducing the two main model classes in this thesis, Gaussian processes (Section 1.4) and variational autoencoders (Section 1.6).

1.1 PROBABILISTIC MACHINE LEARNING

This section introduces the basic concepts in probabilistic machine learning relevant to this thesis, with an emphasis on probabilistic modelling and inductive biases.

In the probabilistic framework we describe our beliefs about the world and our models using probability theory. By writing down a probabilistic model, we express the connections between the known knowns, i.e. observations, and known unknowns, such as latent structures, latent variables, or random noise.

We can then use the rules of inverse probability to infer likely values or whole distributions for unknown quantities from known quantities. Typically, the known quantities are observations at certain locations of the input space – here simply termed "data" – whereas unknown quantities can be model parameters, latent variables, the model structure, or predictions at yet unobserved locations of the input space – in the following also termed "hypothesis". The main tool in this context is Bayes' rule, which can be derived from the sum and the product rule of probability and can abstractly be expressed as (e.g. MacKay (2003)):

Bayes' rule

$$p(\text{hypothesis} \mid \text{data}) = \frac{p(\text{data} \mid \text{hypothesis}) \times p(\text{hypothesis})}{\sum_{\text{h}} p(\text{data} \mid \text{h}) p(\text{h})}$$

More formally, we can express Bayes' rule in terms of observed data $\mathcal{D}$ and model parameters θ for a fixed model m

$$p(\theta \mid \mathcal{D}, m) = \frac{p(\mathcal{D} \mid \theta, m) \times p(\theta \mid m)}{p(\mathcal{D} \mid m)} \tag{1.1}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}.$$

The unknown *posterior* appears on the left hand side of Equation (1.1), whereas all quantities that are known in principle appear on the right hand side of Equation (1.1). The *prior* expresses our belief about how likely certain hypothesis are before having observed any data; the *likelihood* expresses how likely the observed data are under a given hypothesis. The *evidence* acts as a normaliser for the posterior,

$$p(\mathcal{D} \mid m) = \int p(\mathcal{D}, \theta \mid m) \, \mathrm{d}\theta = \int p(\mathcal{D} \mid \theta, m) p(\theta \mid m) \, \mathrm{d}\theta, \tag{1.2}$$

and plays a crucial role in model selection and training of model parameters. It is also referred to as the *marginal likelihood* and is the probability of the data integrated over all hypotheses. We further discuss the marginal likelihood in Section 1.2.

While all quantities within the evidence $p(\mathcal{D} \mid m)$ – the prior and the likelihood – are known, it is usually intractable to perform the sum or integral to marginalise over all the hypotheses or model parameters θ. We therefore have to resort to different approximation techniques in order to perform *approximate inference* in these intractable models, see Section 1.3.

We can view Bayes' rule as an update rule for the prior; our initial belief as expressed by the prior distribution is updated by observations through the likelihood. The posterior distribution then represents our updated belief about the world; it effectively acts as new prior that can be further updated by future observations. We can then use the posterior to make probabilistic predictions at new locations $\mathcal{D}^*$

$$p(\mathcal{D}^* \mid \mathcal{D}, m) = \int p(\mathcal{D}^* \mid \theta, m) p(\theta \mid \mathcal{D}, m) \, \mathrm{d}\theta \tag{1.3}$$

In general, the posterior will not be of the same functional form as the prior, such that each Bayesian update gives rise to a more complex posterior distribution.

conjugate models

A special case arises when the prior and likelihood are *conjugate*, that is, they are such that the posterior is of the same functional form as the prior – in this case, Bayesian updates correspond to simple updates of the parameters of the prior

distribution. Conjugate distributions exist when the likelihood is a member of the exponential family (e. g. Murphy (2012)).

When faced with a particular task or problem, the two main steps in probabilistic modelling and machine learning are to (i) write down a probabilistic model that connects known and unknown quantities and (ii) perform exact or approximate inference in this model (Ghahramani, 2013).

probabilistic model

To write down a probabilistic model m, we first choose the random variables of interest that denote known and unknown quantities, such as observables, latent variables, or model parameters. To write down the "joint probability distribution of everything" (MacKay, 2003) we then specify how these variables are connected to each other. This model structure is often depicted through a *graphical model* that expresses conditional dependencies and independencies and details how the joint distribution of all variables factorises. Finally, we need to specify the particular distributions of each factor, that is, the parametric or non-parametric form of the distributions as well as their parameterisation.

graphical model

In an abstract example of a model with data $\mathcal{D}$ and parameters θ, the joint probability distribution of the data and the model parameters, $p(\mathcal{D}, \theta \mid m)$ factorises into a likelihood $p(\mathcal{D} \mid \theta, m)$ and a prior of the parameters $p(\theta \mid m)$. While the likelihood incorporates observations and is data-dependent, the prior should, in principle, only encode our prior beliefs and be data-independent. As a concrete example, consider a Bayesian neural network (BNN) (Neal, 1994; MacKay, 1995) for regression, where the likelihood is Gaussian with mean expressed through a neural network with individual weights θ on which we place a prior.

Both of these design choices – the model structure and the individual distributions, most notably the priors – fundamentally influence the properties and performance of our model and its ability to learn and generalise from data. Crucially they encapsulate the *inductive biases*, by which we mean "biases for choosing one generalization [sic] over another, other than strict consistency with the observed training instances" (Mitchell, 1980). In other words, in principle, a large number of possible generalisations are consistent with a set of observations such that we need to make further assumptions to reduce them to a small and plausible set. In the introduction we mentioned the universal approximation theorem that states that a single layer feed-forward network of sufficient size can realise any continuous function. In the above example of a BNN, the prior on the individual network weights induces a distribution of "typical" functions that the BNN can express. By altering the network architecture (layers, number of units, weight-sharing, non-linearities, etc.) or the prior on the weights, certain

inductive biases

functions become more or less likely. However, because we place a prior on the individual weights, it is hard to characterise this typical set. In contrast, the covariance function of a Gaussian process prior specifies global properties of typical function draws such as smoothness, periodicity or stationarity. This property makes Gaussian process models particularly amenable for Bayesian modelling, especially when we have expert knowledge about typical functions that we would like to encode. For example, Chapter 4 describes the construction of a GP covariance function that respects general invariances – a property that would be hard to encode in neural networks. Interestingly, in the limit of infinite width a single layer BNN with independent Gaussian prior on the weights converges to a Gaussian process with a particular non-stationary covariance function (Neal, 1994; Williams, 1998).

prior

To reiterate, once the model structure or model class is fixed, the prior expresses our *a priori* belief about how likely each realisable model configuration is. While the likelihood term can up- or down-weight the probability of each of these configurations, it cannot recover configurations that have zero probability under the prior. It is therefore important to make the prior sufficiently broad.

The prior itself may often contain hyperparameters λ, such as the parameters of the Gaussian process covariance function or the scale parameters of a Gaussian prior on the weights of a BNN. In a fully Bayesian treatment, we would also need to infer the values of these parameters through Bayesian inference, which requires a *hyperprior* on λ. This can lead to a cascade of higher and higher order priors (Murphy, 2012). The influence of these parameters often diminishes such that the model becomes increasingly insensitive to them, so this hierarchy is usually truncated at the first level. The hyperparameters λ are then learned or optimised using the objective function directly; this approach is often referred to as *type II maximum likelihood learning* (Rasmussen and Williams, 2006) or *empirical Bayes*. As in conventional maximum likelihood learning, this can lead to overfitting.

type II maximum likelihood

Many tasks within machine learning can be divided into *supervised* and *unsupervised* tasks. In supervised learning, we aim to predict some quantity $y \in \mathcal{Y}$ given an input $\mathbf{x} \in \mathcal{X}$ from a limited number of N training examples $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$. The held-out data at which we evaluate our models performance is usually referred to as the test set $\mathcal{D}^* = \{\mathbf{x}_n^*, y_n^*\}_{n=1}^{N^*}$.

supervised learning

In unsupervised learning, the training and test sets only consists of unlabelled observations or targets, $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$ and $\mathcal{D}^* = \{\mathbf{x}_n^*\}_{n=1}^{N^*}$ respectively, and the tasks are much more diverse. They range from clustering data into groups, learning representations, compressing data, uncovering hidden structure, or building a density model. In general, unsupervised learning models contain

unsupervised learning

latent structures and variables, such as cluster centroids and assignments, latent codes, or topic membership, that we wish to infer.

Supervised learning tasks can be further divided into *regression* and *classification* tasks. In regression tasks the outputs y_n are modeled as continuous functions of the inputs $\mathbf{x}_n$ and can be single or multi dimensional. A typical likelihood function in a probabilistic regression model is Gaussian observation noise ϵ on top of a linear or non-linear transformation of the inputs, $f(\mathbf{x})$,

regression

$$y = f(\mathbf{x}) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)\,. \tag{1.4}$$

$f(\mathbf{x})$ is also referred to as the *regression function*.

In contrast to regression, the outputs in classification tasks are typically discrete and are referred to as *labels*; every input is categorised into one out of C possible classes. Typical likelihoods for classification are the Bernoulli likelihood (for binary classification) or the Categorial likelihood (for more than two classes). These likelihoods can be expressed through the softmax function

classification

$$p(y = c \mid \mathbf{x}, \theta) = \frac{\mathrm{e}^{\Phi_\varphi(\mathbf{x})^\intercal \mathbf{w}_c}}{\sum_{c'} \mathrm{e}^{\Phi_\varphi(\mathbf{x})^\intercal \mathbf{w}_{c'}}}, \tag{1.5}$$

where $\Phi_\varphi(\mathbf{x})$ is a linear or non-linear transformation (with parameters φ) of the inputs including a constant output to model biases, $\mathbf{w}_c$ denotes the weights for class c, and θ denotes the collection of all parameters. $\Phi_\varphi(\mathbf{x})$ is also referred to as *feature representation* of the input $\mathbf{x}$.

In the case of binary classification, the softmax function in Equation (1.5) can be simplified to a sigmoid function σ and the likelihood is given by

$$p(y = c_1 \mid \mathbf{x}, \theta) = \sigma(\Phi_\varphi(\mathbf{x})^\intercal \mathbf{w}_1) \tag{1.6}$$

$$p(y = c_2 \mid \mathbf{x}, \theta) = 1 - \sigma(\Phi_\varphi(\mathbf{x})^\intercal \mathbf{w}_1) \tag{1.7}$$

Orthogonal to the division into supervised and unsupervised learning tasks, machine learning models can be divided into *parametric* and *non-parametric* models.

The hallmark of parametric methods is that the involved regression functions, feature representations, or distributions are explicitly parameterised in terms of functions with a fixed, finite number of parameters. Examples of parametric methods range from simple linear or logistic regression to highly complex deep neural networks. While the capacity and expressiveness of these methods are very different, a common factor is that the capacity stays the same regardless of the size of the training data. This has the advantage of fixed computational cost but the disadvantage of not being able to extend capacity when data at new input locations is observed.

parametric methods

Bayesian non-parametrics

In contrast, the capacity of (Bayesian) non-parametric models grows with the training data size. However, this can lead to potentially prohibitive computational costs for large data. Parts of this thesis are concerned with Gaussian processes, a prominent class of Bayesian non-parametrics, which we introduce in Section 1.4. For an excellent recent survey on Bayesian non-parametrics refer to Ghahramani (2013).

1.2 THE MARGINAL LIKELIHOOD

The evidence or marginal likelihood plays a crucial role in model selection and training of probabilistic models. This section elaborates on both these aspects and explains the built-in inductive biases for model selection: a Bayesian version of Occam's razor. We use the marginal likelihood as a starting point to choose and optimise our models throughout this thesis.

maximum likelihood learning

Most current non-probabilistic machine learning models are trained by maximising the likelihood $p(\mathcal{D} \mid \theta, m)$ with respect to model parameters θ directly. One issue with this objective function is that it does not distinguish between models which fit the training data equally well but will have different generalisation characteristics. In an extreme case, some models may fit the training data better but generalise worse or not at all; when this happens, this is typically known as *overfitting*. Some form of regularisation is often employed to avoid said overfitting, improve generalisation, or encourage other model properties such as sparsity. Nonetheless, it is usually difficult to assess these from the likelihood alone and validation sets or cross-validation techniques must be employed (Bishop, 2006).

overfitting

The archetypal example of such a case is one of polynomial regression (see, for example, Rasmussen and Ghahramani (2001), Bishop (2006) and MacKay (2003)). In polynomial regression, models differ by the order of polynomials used as regression function to explain the data. In all cases, the learnable parameters θ are given by the various coefficients. Typically, three (or more) cases are considered: (i) a very low order polynomial m_1 that is not flexible enough to fit the training data well; (ii) an intermediate order polynomial m_2 that is just flexible enough; (iii) and a very high order polynomial m_3 that perfectly fits the training data for a very fine-tuned set of coefficients that typically lead to a strongly oscillating function.

When only considering the (training) likelihood, m_3 will perform best and m_1 will perform worst; however, we expect m_2 to generalise the best, followed by m_1 and then m_3. The problem is that the likelihood does not take into account the *complexity* of the regression functions. Intuitively, we expect the model that generalises best to be complex enough to fit the data, but not so complex as to

overfit and model random noise. This heuristic is referred to as *Occam's razor* and is discussed in the context of machine learning by Jefferys and Berger (1992) and Rasmussen and Ghahramani (2001). More generally Occam's razor states that out of competing methods with same training set fit, one should prefer the one with fewest assumptions. By integrating the likelihood against the prior, the marginal likelihood takes complexity into account and penalises more complex models over simpler ones (Rasmussen and Ghahramani, 2001; MacKay, 2003; Rasmussen and Williams, 2006). The marginal likelihood is also closely related to bounds on the generalisation error (Seeger, 2003; Germain et al., 2016).

Occam's razor

Intuitively, this complexity penalty is a volume argument: complex models (e.g. higher order polynomials) spread their prior mass over a larger space than simpler models (e.g. lower order polynomials); and while a particular fine-tuned configuration of a complex model may fit the training data extremely well, this configuration occupies a very small volume compared to the likely configurations of a simpler model. At the same time, an overly simplistic model might not be able to fit the data well at all. Choosing the right model is a matter of evaluating this trade-off. Rasmussen and Ghahramani (2001) make this argument more precise for several simple regression examples. They evaluate the marginal likelihood for different models of varying complexities and confirm that it rewards models with just enough but not too much complexity. Gaussian process models for regression and classification also allow for the marginal likelihood or good approximations of it to be evaluated (see Chapters 2 and 3) and are therefore accessible to a similar analysis. In Section 3.1 we revisit the volume argument for the objective function of sparse Gaussian processes. In Chapter 4 we use Bayesian model selection and the log marginal likelihood to identify and learn invariances.

In summary, the marginal likelihood can be used to compare different models by applying Bayes' rule at the level of the models instead of the parameters (e.g. Ghahramani (2015) and MacKay (2003)):

Bayesian model selection

$$p(m \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid m)p(m)}{p(\mathcal{D})} \propto p(\mathcal{D} \mid m)p(m)\,. \tag{1.8}$$

1.3 VARIATIONAL INFERENCE

Exact inference is only possible in the simplest probabilistic models. As we alluded to above, in most cases the marginal likelihood is intractable, and solving the inference problem corresponds to computing the intractable high-dimensional evidence integral (Equation (1.2)). Fortunately, many approximation techniques have been developed that allow us to perform *approximate inference* in a growing class of models. Most approaches fall in one of the following categories: Markov Chain Monte Carlo (MCMC) methods (Neal, 1993, 2011), variational inference

approximate inference

(VI) (Jordan et al., 1999; Wainwright and Jordan, 2007), expectation propagation (EP) (Minka, 2001), or sequential Monte Carlo (SMC) (Doucet et al., 2000).

In this thesis, we focus on variational methods and therefore only introduce variational inference in detail. For the other approximation techniques, we refer the reader to the references above. Moreover, we focus on aspects of variational inference that are directly relevant to this thesis and refer the reader to Jordan et al. (1999) and Wainwright and Jordan (2007) for a detailed exposition, MacKay (2003) for a light introduction from a physics point of view, and Blei et al. (2017) for an excellent recent review.

Variational inference

Variational inference transforms an intractable integration problem into an optimisation problem. To do this, the intractable distribution $p(z)$ is replaced by a flexible but usually simpler distribution $q_\varphi(z)$, for which the integration problem can be solved efficiently. We then aim to make q_φ as close to p as possible by adapting its *variational parameters* φ. Formally, this is done by minimising the *KL-divergence* or relative entropy from q_φ to p:

KL-divergence

$$\mathrm{KL}(q_\varphi(z) \parallel p(z)) = \int q_\varphi(z) \log \frac{q_\varphi(z)}{p(z)} \,\mathrm{d}z \geq 0 \tag{1.9}$$

The KL divergence is asymmetric, always non-negative, and zero if and only if q_φ and p are equal. Its second order Taylor approximation is symmetric by design and the corresponding Hessian matrix is referred to as the *Fisher information matrix*, which plays a crucial role in the field of *information geometry* (Amari, 2016). It can be used to define a distance metric between distributions as well as *natural gradients*, which can lead to drastically faster convergence rates for gradient descent based algorithms; refer to Martens (2014) for a recent review.

In practice, the approximating family of distributions $\{q_\varphi\}_\varphi$ often does not contain the intractable distribution $p(z)$, such that all solutions obtained even with the optimal q^*_φ are suboptimal compared to the true distribution $p(z)$. The resulting difference to the true solution is referred to as the *approximation gap* (Cremer et al., 2018).

approximation gap

A common simple choice for q_φ is the Normal distribution which often allows us to compute integrals and moments in closed form. The variational parameters then correspond to its mean μ and covariance matrix Σ. In many cases the covariance is constrained to be diagonal or even isotropic to reduce the number of learnable parameters.

Variational inference can be applied both to global variables that affect all datapoints, such as the weight posteriors of a BNN (Blundell et al., 2015) or the inducing inputs of sparse Gaussian process approximations (Titsias, 2009a), as

well as local per-datapoint variables, such as the approximate posterior in a latent variable model (see Section 1.5). In the latter case, the model contains one or more variational parameters *per datapoint* that must be optimised.

This has several, usually negative, consequences which preclude scaling to larger datasets. The two main computational ramifications are: (i) the number of (local) variational parameters grows as the size of data set grows; (ii) after training the model, the (local) variational parameters for new test points still need to be optimised during test time. A solution to both these problems is *amortised* variational inference; instead of optimising the per-datapoint variational parameters directly, we train a so-called *inference*, *recognition*, or *encoder* network that outputs the variational parameters. The inference network has a fixed size and allows for fast test-time inference through a single forward pass. However, the resulting variational parameters may not be optimal for that particular datapoint. This suboptimality is usually referred to as *amortisation gap* (Cremer et al., 2018). In Section 1.6 we introduce a family of deep generative models called variational autoencoders (P. and Welling, 2014; Rezende et al., 2014), which popularised amortised variational inference.

amortised inference

amortisation gap

1.4 GAUSSIAN PROCESS METHODS

The first part of this thesis, Chapters 3 and 4, presents analyses and advances in Gaussian process models. In the following, we therefore provide an exposition of Gaussian processes for supervised learning with a focus on regression. We touch on classification and unsupervised learning with Gaussian processes in Sections 1.4.3 and 1.5, respectively. For a comprehensive textbook on Gaussian processes in machine learning refer to Rasmussen and Williams (2006)

1.4.1 *Inference in Gaussian processes for regression*

Given a training dataset $\mathcal{D}$ of N observations of input/output pairs, $\mathcal{D} = \{(\mathbf{x}_n, y_n) \mid n = 1, \dots, N\}$, we wish to make predictions y^* at previously unobserved test inputs $\mathbf{x}^*$. We collect the training inputs $\mathbf{x}_n$ into a matrix X and the (possibly noisy) training outputs y_n into a vector $\mathbf{y}$. Typically, $\mathbf{x}$ is multi-dimensional and continuous, whereas y is one-(or low-) dimensional and can be either continuous (in case of regression) or discrete (in case of classification). In the following, we use a one-dimensional dataset as an illustrative example, see Figure 1.1.

In GP regression we model the outputs y by an (unobserved) *latent function* f, which returns predictions for all possible new inputs $\mathbf{x}^*$, and Gaussian noise, see

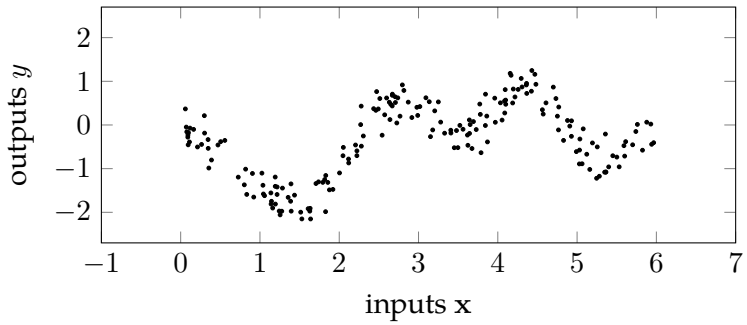


Figure 1.1: Example regression problem from Snelson and Ghahramani (2006) for GP regression with one-dimensional inputs $\mathbf{x}_n$ and noisy one-dimensional outputs y_n.

Equation (1.4). With these assumptions, exact posterior inference is fully tractable as we discuss in the following.

Formally, "a Gaussian process is a collection of random variables, any finite number of which have a consistent[1] joint Gaussian distribution" (Rasmussen and Williams, 2006, Definition 2.1). It is fully determined by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. The covariance function encodes our assumptions and inductive biases on the latent function f, such as smoothness, amplitude, periodicity, or other symmetries. Intuitively, it defines a notion of similarity of inputs. When the covariance between two inputs is large, the corresponding function values co-vary. When it is small, the function values vary independently. For now, we assume the covariance function is given and discuss its choice in more detail in Section 1.4.2.

As stated above, we model the entire latent function f by a GP prior. By definition, a finite collection of latent function values at training inputs X and future test inputs X^* is assumed to have a joint Gaussian distribution:

$$f(\cdot) \sim \mathcal{GP}(0, k(\cdot,\cdot)) \qquad \curvearrowright \qquad p(\mathbf{f}, \mathbf{f}^*) = \mathcal{N}\left(0, \begin{bmatrix} K_{\mathbf{ff}} & K_{\mathbf{f}*} \\ K_{*\mathbf{f}} & K_{**} \end{bmatrix}\right) \tag{1.10}$$

where $\mathbf{f}$ and $\mathbf{f}^*$ denote the latent function values at the training inputs X and test inputs X^*, respectively, and we have introduced the covariance matrices $[K_{\mathbf{ff}}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $[K_{*\mathbf{f}}]_{ij} = k(\mathbf{x}^*_i, \mathbf{x}_j)$, and so on. For simplicity we have assumed the data to be zero-centred and stationary such that a zero mean function is appropriate. The training observations $\mathbf{y}$ at the input locations are assumed to be

1 By "consistent" we mean that "the random variables obey the usual rules of marginalisation, etc." (Quiñonero-Candela and Rasmussen, 2005).

noisy observations around the latent function values $\mathbf{f}$ and are incorporated via the Gaussian likelihood $p(\mathbf{y} \mid \mathbf{f})$:

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_{n=1}^{N} \mathcal{N}\left(y_n; f_n, \sigma_n^2\right) \tag{1.11}$$

where the scalar σ_n^2 is referred to as *noise variance*. In the noise free limit, $\sigma_n^2 \to 0$, the likelihood collapses to a delta-function at $\mathbf{y} = \mathbf{f}$. Other noise models are, of course, possible; however, to obtain analytically tractable expressions a conjugate Gaussian likelihood is often used.

noise variance

We obtain the joint posterior over the latent function values from Bayes' rule:

$$p(\mathbf{f}, \mathbf{f}^* \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f}, \mathbf{f}^*)}{p(\mathbf{y})} \tag{1.12}$$

As the prior and the likelihood are Gaussian, exact Bayesian inference is possible and the posterior predictive probability of $\mathbf{f}^*$ is again given by a GP. It is obtained by marginalising over the latent function values $\mathbf{f}$:

$$p(\mathbf{f}^* \mid \mathbf{y}, X, X^*) = \int p(\mathbf{f}, \mathbf{f}^* \mid \mathbf{y}, X, X^*) \, \mathrm{d}\mathbf{f} = \mathcal{N}\left(\mathbf{f}^* \mid \boldsymbol{\mu}^*, \Sigma^*\right) \tag{1.13a}$$

$$\boldsymbol{\mu}^* = K_{*\mathbf{f}}[K_{\mathbf{ff}} + \sigma_n^2 I]^{-1}\mathbf{y} \tag{1.13b}$$

$$\Sigma^* = K_{**} - K_{*\mathbf{f}}[K_{\mathbf{ff}} + \sigma_n^2 I]^{-1}K_{\mathbf{f}*} \tag{1.13c}$$

Observe that the predictive mean is linear both in the observations $\mathbf{y}$ and in the covariances with the test inputs, $K_{*\mathbf{f}}$. The predictive covariance at the test points, Σ^* is the original full covariance K_{**} minus the conditional covariance that can already be explained by the training data $\mathcal{D}$. Figure 1.2 illustrates the posterior predictive distribution of exact GP inference for our illustrative example.

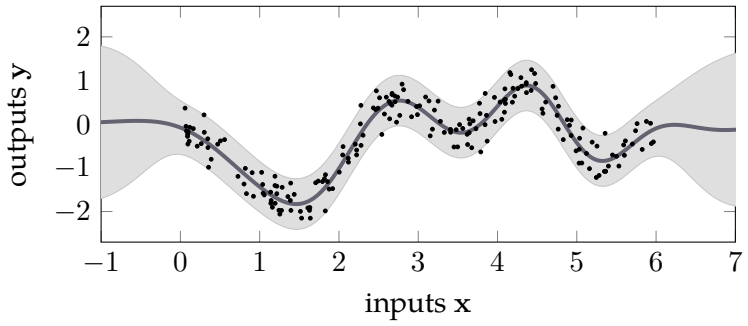


Figure 1.2: Posterior predictive distribution of exact GP inference with a squared exponential kernel and learned hyperparameters. We plot the predictive mean (grey line) and two standard deviations (shaded area).

The log of the marginal likelihood is analytically tractable and obtained by marginalising over the latent regression function f:

$$\log p(\mathbf{y} \mid X) = \log \int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f}, \mathbf{f}^*) \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{f}^* \tag{1.14}$$

$$= -\frac{1}{2}\mathbf{y}^\intercal (K_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log \left| K_{\mathbf{ff}} + \sigma_n^2 \right| - \frac{N}{2}\log 2\pi. \tag{1.15}$$

As discussed in Section 1.2 we can use the marginal likelihood for model comparison to, for example, select suitable priors or tune the noise variance σ_n^2.

1.4.2 *Covariance functions and hyperparameter learning*

So far, we have assumed the covariance function $k(\mathbf{x}, \mathbf{x}')$ to be given. But how should we choose it in practice? This usually depends on the task at hand; there exists an abundance of literature about this topic, for example, Rasmussen and Williams (2006, Chapter 4) and Duvenaud (2014).

kernel

The covariance function plays the same role as the *kernel* in Support Vector Machines (SVM); refer to Schölkopf and Smola (2002) and Steinwart and Christmann (2008) for two comprehensive books on SVMs and Stulp and Sigaud (2015) for an excellent review relating GPs to SVMs and other widely used (kernalised and non-kernalised) regression algorithms. Due to this close connection we use the terms covariance function and kernel function, or kernel for short, interchangeably.

As mentioned above, choosing a particular covariance function determines the properties of the latent functions that can be represented by the GP prior. Observations then restrict these functions further through their likelihood to, for example, pass through or close to certain points or have a certain derivative at a particular location.

squared exponential kernel

A very popular choice for the covariance function in GPs is the squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = s_f^2 \exp(-\tfrac{1}{2}|\mathbf{x} - \mathbf{x}'|^2/\ell^2) \tag{1.16}$$

kernel and model hyperparameters

where the signal variance s_f^2 and the lengthscale ℓ are the *kernel hyperparameters*. Together with the noise variance σ_n^2 of the likelihood they form the vector $\boldsymbol{\theta}$ of *model hyperparameters*. The signal variance determines the scale on which the latent function f varies, whereas the lengthscales determines how quickly it changes.

To make the functions invariant to finite shifts, that is, translations by a period T, we can include trigonometric functions, such as cosines, in the covariance function (MacKay, 1998); inputs a period apart are then maximally correlated.

In Chapter 4 we construct a covariance function that can encode arbitrary general symmetries or invariances and can be used to learn them using the marginal likelihood.

In general, new kernels can be constructed out of existing kernels (Rasmussen and Williams, 2006, Section 4.2.4): If k_1 and k_2 are valid covariance functions on a domain $\mathcal{X}$, so are $k_1 + k_2$ as well as $k_1 \times k_2$. Moreover, if Φ is a mapping between two sets, $\Phi : \widetilde{\mathcal{X}} \twoheadrightarrow \mathcal{X}$, and k is a valid kernel on $\mathcal{X}$ then $k(\Phi(\mathbf{x}), \Phi(\mathbf{x}'))$ is a valid kernel on $\widetilde{\mathcal{X}}$. For example, by combining a squared exponential kernel and a periodic kernel, we can build a prior for functions with modulated periodicity.

Choosing the "right" covariance function is a highly non-trivial task. Fortunately, the marginal likelihood can be used to select the best model or covariance function out of several candidates. However, even enumerating the covariance functions that can be constructed by combining simpler kernels is already a combinatorial problem. Thus, a naive approach is infeasible. Several recent approaches attempt to automatically infer *one* good covariance function from the data, such as the automatic statistican (Steinruecken et al. (2019) provide a recent summary) or kernel discovery through latent space optimisation (Lu et al., 2018). In both of these cases, richer kernels are constructed out of simpler ones. Orthogonal to this, approaches such as deep kernel learning (DKL) (Wilson et al., 2016b) use deep neural networks to extract features from the inputs and apply a simple kernel to them.

Selecting the functional form of the kernel is only the first part of specifying the covariance function. We must also identify the best model hyperparameters. As discussed in Section 1.1 there are two basic ways to set the hyperparameters of a probabilistic model: (i) a fully Bayesian treatment: place a hyperprior on the hyperparameters and then perform Bayesian inference on them as well; (ii) type II maximum likelihood learning: directly optimise the marginal likelihood w.r.t. $\boldsymbol{\theta}$. While (i) is more principled, it is also often infeasible. In practice, one frequently resorts to (ii) and chooses the hyperparameters that optimise the log marginal likelihood. However, this is usually a non-convex optimisation problem with local minima and saddle points. Moreover, it may lead to overfitting (refer to the discussion in Section 1.4.4 and Chapter 3) or, indeed, underfitting; for example, when GPs are fit to a very small number of points in this way they often over-estimate the lengthscale.

Both approaches for choosing hyperparameters in a principled way constitute one major advantage of GPs over other kernel methods; for the latter, there is no such clear cut objective which often necessitates cross-validation or learning the kernel matrix (Bousquet and Herrmann, 2003).

1.4.3 *Gaussian process models for classification*

So far we have explained Gaussian process regression with a Gaussian likelihood function. In classification, the outputs y are no longer continuous but discrete – every input $\mathbf{x}$ belongs to one out of C possible classes. Here, we briefly discuss Gaussian process classification. Fore a more in-depth introduction see Rasmussen and Williams (2006, Chapter 3).

A first approach is to ignore the discreteness of the classes and model the label with a Gaussian likelihood regardless. We take this approach in parts of Chapter 4 as our estimators of the objective there are only unbiased for Gaussian likelihoods. Using a Gaussian likelihood has the advantage that we can use the machinery developed for regression; however, it has several disadvantages. Foremost, our modelling assumptions are clearly violated, which can lead to poor predictive performance. Second, our predictive uncertainty becomes meaningless. Moreover, especially when modelling more than two classes with this approach, we implicitly assume an ordering of the classes that is not founded in reality – 1 would be closer to 2 than to 4, for example. Though, modelling several one-hot encoded outputs with Gaussian likelihoods instead can alleviate this problem and often works well in practice.

The more principled approach uses classification likelihoods (Rasmussen and Williams, 2006). In the case of binary classification, we place a GP prior on the latent function $f(\mathbf{x})$, which is subsequently squashed by a logistic function to model the prior distribution

$$\pi(\mathbf{x}) = p(y = 1 \mid f(\mathbf{x})) = \sigma(f(\mathbf{x}))\,. \tag{1.17}$$

Similarly to the regression case, inference consists of computing the posterior distribution of the latent function f^* at a new input $\mathbf{x}^*$ and integrating it over the likelihood:

$$p(f^* \mid X, \mathbf{y}, \mathbf{x}^*) = \int p(f^* \mid X, \mathbf{x}^*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) \,\mathrm{d}\mathbf{f} \tag{1.18}$$

$$p(y^* = 1 \mid X, \mathbf{y}, \mathbf{x}^*) = \int \sigma(f^*) p(f^* \mid X, \mathbf{y}, \mathbf{x}^*) \,\mathrm{d}f^*\,. \tag{1.19}$$

Because of the non-conjugate likelihood, inference is analytically intractable and we must apply approximations. In Section 2.5 we describe a variational approach for large scale classification.

1.4.4 *Notes on implementations and libraries*

To compute the marginal likelihood in Equation (1.14) and make predictions, we must compute the inverse and log-determinant of the kernel matrix $K_{\mathbf{ff}} + \sigma_n^2 I$. For numerical stability these computations are performed using the Cholesky decomposition, $K_{\mathbf{ff}} + \sigma_n^2 I = LL^\intercal$, where L is a lower triangular matrix. Moreover, kernel matrices without noise, such as $K_{\mathbf{ff}}$ by itself, can become singular, such that their inverse might not exist. In such cases, a small diagonal *jitter*-term ϵ is added to ensure non-singularity. We will discuss the influence of this term further in Chapter 3.

The optimisation of the marginal likelihood w.r.t. the hyperparameters is often performed using second order gradient methods such as L-BFGS (Liu and Nocedal, 1989). For these, the gradients of the objective w.r.t. the hyperparameters are needed. For their computation and, in fact, many of the computations associated with GPs, *The Matrix Cookbook* (Petersen and Pedersen, 2012) is an invaluable resource. For further details on the implementation of GPs, refer to the documentation of standard toolboxes such as `gpml` for `matlab` (Rasmussen and Nickisch, 2010), `GPy` for `python` (The GPy authors, 2012), or `GPflow` (Matthews et al., 2017) for `python` using `tensorflow` (Martín Abadi et al., 2015).

All exact implementations of fully general GPs scale as $\mathcal{O}(N^3)$ in time and $\mathcal{O}(N^2)$ in memory complexity, where N is the number of training inputs. This is due to the computation of the log-determinant and the inverse of the covariance matrix; computing the eigenvalues and the Cholesky decomposition takes $\mathcal{O}(N^3)$. This limits exact GP inference to small to medium sized datasets with on the order of 10,000 data points in many ways.

A growing number of approximate inference techniques allow us to scale Gaussian process inference beyond these limitations. In Chapter 2 we introduce several of these approximation techniques and analyse two of them in depth in Chapter 3.

1.5 CONTINUOUS LATENT VARIABLE MODELS

So far we have described supervised learning methods, which map inputs $\mathbf{x}$ to outputs y. A second pillar of machine learning is *unsupervised learning*, in which we aim to model inputs $\mathbf{x}$ or their density $p_{\text{Data}}(\mathbf{x})$, usually by means of some form of latent structure or variables. Often these help us to interpret the data without labels and uncover its structure.

In the second part of this thesis we discuss the priors in variational autoencoders (VAEs), which are an example of continuous latent variable models with a

particular inference scheme. This section introduces continuous latent variable models in general before we discuss VAEs in more detail in Section 1.6.

Continuous latent variable models (LVMs) are one type of unsupervised probabilistic model that explain observables $\mathbf{x}$ through unobserved continuous latent variables $\mathbf{z}$. More specifically, we use continuous latent variable models to model a high-dimensional data distribution $p_{\text{Data}}(\mathbf{x})$ by means of a simple prior distribution $p_\lambda(\mathbf{z})$ in a latent space, which is warped by a likelihood model $p_\theta(\mathbf{x} \mid \mathbf{z})$ to fit the data, see Figure 1.3.

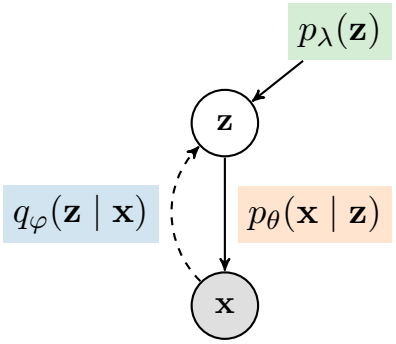


Figure 1.3: Graphical model of a generic latent variable model with prior $p_\lambda(\mathbf{z})$ and likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$. To perform *(amortised) variational inference*, we also include a parametrised inference distribution $q_\varphi(\mathbf{z} \mid \mathbf{x})$ also referred to as *approximate posterior*.

Formally, a LVM is defined through a joint probability of observed and unobserved variables $p_\theta(\mathbf{x}, \mathbf{z})$. Using the product rule of probability, this joint probability is factorised into a prior $p_\lambda(\mathbf{z})$ and a likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$,

$$p_{\theta,\lambda}(\mathbf{x}, \mathbf{z}) = p_\lambda(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z}), \tag{1.20}$$

where λ and θ denote the free parameters of the prior and likelihood, respectively. Thus, we can define a latent variable model by specifying the prior on the latent codes and the data likelihood under those codes. The latent variables $\mathbf{z}$ can be both discrete and continuous; here, we focus on the latter. Usually, we choose the prior to be a standard Normal distribution, $p(\mathbf{z}) = \mathcal{N}(0, \mathbb{1})$.

The model distribution is then given by the marginal likelihood

$$p_\theta(\mathbf{x}) = \int p(\mathbf{z})\, p_\theta(\mathbf{x} \mid \mathbf{z})\, \mathrm{d}\mathbf{z}, \tag{1.21}$$

1.5.1 *Examples of continuous latent variable models*

Three common models for the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ are:

LINEAR GAUSSIAN MODEL

A Gaussian likelihood model with linear mean function and isotropic covariance corresponds to probabilistic principal component analysis (PCA) (Tipping and Bishop, 1999; Roweis, 1998):

$$\mathbf{x} = W\mathbf{z} + \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbb{1}) \tag{1.22}$$

In this case, the model allows for an exact closed-form solution, which is defined up to arbitrary orthogonal rotations of the latent space (Tipping and Bishop, 1999). Probabilistic PCA is closely related to factor analysis (Spearman, 1904), which assumes diagonal but not necessarily isotropic covariance; factor analysis no longer allows for a closed-form solution of the weights W but iterative methods exist (Rubin and Thayer, 1982).

NON-LINEAR GAUSSIAN PROCESS MODEL

A simple yet flexible non-linear model is the Gaussian Process Latent Variable Model (GP-LVM) (Lawrence, 2004), which uses a GP to model the mean of a Gaussian likelihood with isotropic covariance:

$$\mathbf{x} = \mathbf{f}(\mathbf{z}) + \boldsymbol{\epsilon} \qquad \mathbf{f} \sim \mathcal{GP}(\boldsymbol{\mu}, K); \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbb{1}) \tag{1.23}$$

As the prior distribution on the latent space is warped through a non-linear function, exact inference is impossible. Titsias and Lawrence (2010) proposed a variational approach to obtain a lower bound to the marginal likelihood, following earlier works on variational inference for GP regression (Titsias, 2009a).

NON-LINEAR NEURAL NETWORK MODEL

Instead of GPs we can also parametrise the likelihood function through a neural network. Besides Gaussian noise models, a range of likelihood distributions is used in practice, such as the Bernoulli distribution but also more complicated auto-regressive structures or the discretised mixture of logistics likelihood (Salimans et al., 2017). For these more complicated likelihoods we not only parameterise the means but also the other parameters of the likelihood, such as the scales or mixture components.

When this model is trained using *amortised variational inference* for the approximate posterior, it is usually referred to as variational autoencoder (VAE) (P. and Welling, 2014; Rezende et al., 2014), though strictly speaking it is not an autoencoder but a deep probabilistic latent variable model. We provide a more detailed introduction to VAEs in Section 1.6 and discuss their priors in Chapter 5.

1.5.2 *Applications of continuous latent variable models*

Continuous LVMs are used for a large number of reasons and applications but also as part of larger machine learning systems. These range from generative modelling and density estimation, unsupervised representation learning and dimensionality reduction to disentanglement and interpretable machine learning. Here, we provide an overview of some of these applications before discussing training and properties of continuous LVMs in more detail.

An early motivation for continuous latent variable models has been dimensionality reduction: While the data distribution might be very high dimensional, it can often be accurately described by much fewer degrees of freedom (Bishop, 2006). In other words, the data distribution often lies on a lower dimensional manifold that is embedded in a higher dimensional space, and we try to map the prior distribution to this complicated lower dimensional manifold through the likelihood. Early non-probabilistic examples for this are principal component analysis (PCA) (Pearson, 1901; Hotelling, 1933; Jolliffe, 1986), whose probabilistic counterparts we mentioned above, and factor analysis (FA) (Spearman, 1904). The GP-LVM can, for example, be used to find a low-dimensional and interpretable periodic structure in character animation (Quirion et al., 2008).

Unsupervised representation learning is probably closest to the motivation given above. It aims to extract meaningful (low dimensional) representations from the data, such as clusters, or structure the latent space, that can be used in downstream tasks. For example, Vylomova et al. (2016) show that differences in the latent space can have semantic meaning for word embeddings. In this case, the prior on the latent space can inform inductive biases that encourage certain structures in the latent space, such as clusters or axis-aligned representations (albeit not necessarily with the standard Normal prior). However, these observations should not be taken to mean that continuous latent variable models necessarily learn Euclidean structures in the latent space. Hauberg (2018) illustrates this by applying input-dependent concentric rotations to the latent space, where the forward and backward rotation become part of the posterior and likelihood, respectively. This rotation leaves the prior density unaltered but completely changes the relational structure of the latent space. Tosi et al. (2014) and Hauberg (2018) suggests to use ideas from Riemannian geometry to pull back the metric in the output space to the latent space to measure distances and define shortest paths.

More recently, Higgins et al. (2017) proposed to alter the objective function of VAEs (discussed below) to encourage axis-aligned or disentangled representations in the latent space for representation learning. In other words, they aim to learn a model such that varying individual dimensions in the latent space lead to

semantically meaningful and interpretable changes in the output space. For example, when modelling faces, one dimension might change the hair colour whereas another influences the age. While much of this work is empirical, Rolinek et al. (2019) and Mathieu et al. (2019) provide first theoretical steps towards understanding these observations. Again, this structure is more accidental than actively encouraged, refer to the earlier discussion of Euclidean distances. Locatello et al. (2019) challenge common assumptions in this task and question whether semantically meaningful changes can be uncovered in a fully unsupervised manor.

Closely related to representation learning are more recent approaches that employ latent space optimisation: Instead of optimising objectives in the high-dimensional and possibly discrete data space, we work with a learned continuous latent representation instead. For example, Gómez-Bombarelli et al. (2018) design molecules with certain properties, Lu et al. (2018) optimise kernels of Gaussian processes, and Rusu et al. (2019) devise a meta-learner that adapts a base learner through optimisation in the latent space.

In generative modelling we care less about the latent structure and more about the model's ability to model the data distribution, for example for interpolation or extrapolation. Tasks can range from inpainting or data imputation to generalisation beyond the training examples. Typical approaches for this task are variational autoencoders (VAEs) (P. and Welling, 2014; Rezende et al., 2014) and extensions thereof both of which we discuss in more detail in Section 1.6. Generative adversarial networks (GANs) (Goodfellow et al., 2014) are a recently introduced, non-probabilistic approach to generative modelling that do not model densities but allow sample generation.

1.5.3 *The marginal likelihood of latent variable models*

To arrive at a trainable model for $\mathbf{x}$, we marginalise out the latent variables $\mathbf{z}$ to obtain the *marginal likelihood* of a latent variable model, similarly to the approach for Gaussian processes

$$p_{\theta,\lambda}(\mathbf{x}) = \int p_\lambda(\mathbf{z})\, p_\theta(\mathbf{x} \mid \mathbf{z})\, \mathrm{d}\mathbf{z}, \tag{1.24}$$

The marginal likelihood defines the model distribution on the observables $\mathbf{x}$. In the following we we drop the subscripts and only re-introduce them when discussing optimisation.

We have discussed why the marginal likelihood is a sensible objective function to train a probabilistic model. Here, we provide an alternative motivation in terms of making the model distribution as close as possible to the empirical data distribution. Let $p_{\mathrm{Data}}(\mathbf{x})$ be the data distribution. Typically, we do not have access

to $p_{\text{Data}}(\mathbf{x})$ but only to samples $\mathcal{D} = \{\mathbf{x}_n\}_n^N$ from it, which define an empirical distribution

$$\widehat{p}_{\text{Data}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\delta(\mathbf{x}-\mathbf{x}_n). \tag{1.25}$$

A good latent variable model matches this empirical distribution well but also generalises to future samples from p_{Data}. To perform inference, or train the model parameters, we therefore minimise the KL divergence between the (empirical) data distribution and the model distribution:

$$\theta^* = \arg\min_{\theta} \text{KL}(\widehat{p}_{\text{Data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})). \tag{1.26}$$

As the data distribution does not depend on the model parameters θ, this is equivalent to maximising the log marginal likelihood on the training set.

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{x\sim\widehat{p}_{\text{Data}}(\mathbf{x})} \log p_\theta(\mathbf{x}). \tag{1.27}$$

The details of how to perform inference in practice, or how to compute $p(\mathbf{z} \mid \mathbf{x})$ or an approximation to it, are model dependent.

1.5.4 *The evidence lower bound (ELBO)*

The log marginal likelihood of a latent variable model is in general intractable, such that we cannot directly optimise it to find the best model hyperparameters[2]. Using variational inference we can obtain a lower bound to the log marginal likelihood of a data point $\mathbf{x}$, by introducing an approximate posterior distribution $q(\mathbf{z} \mid \mathbf{x})$ to the true posterior $p(\mathbf{z} \mid \mathbf{x})$, which can be evaluated more easily:

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{z}\mid\mathbf{x})} && (1.28)\\
&= \log \frac{p(\mathbf{x}\mid\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\mid\mathbf{x})}\frac{q(\mathbf{z}\mid\mathbf{x})}{q(\mathbf{z}\mid\mathbf{x})} && (1.29)\\
&= \int q(\mathbf{z}\mid\mathbf{x}) \log \frac{p(\mathbf{x}\mid\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\mid\mathbf{x})}\frac{q(\mathbf{z}\mid\mathbf{x})}{q(\mathbf{z}\mid\mathbf{x})}\,\mathrm{d}\mathbf{z} && (1.30)\\
&= \underbrace{\int q(\mathbf{z}\mid\mathbf{x}) \log \frac{q(\mathbf{z}\mid\mathbf{x})}{p(\mathbf{z}\mid\mathbf{x})}\,\mathrm{d}\mathbf{z}}_{=\text{KL}(q(\mathbf{z}\mid\mathbf{x})\parallel p(\mathbf{z}\mid\mathbf{x}))} + \underbrace{\int q(\mathbf{z}\mid\mathbf{x}) \log \frac{p(\mathbf{x}\mid\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}\mid\mathbf{x})}\,\mathrm{d}\mathbf{z}}_{=:\mathcal{F}_{\mathbf{x}}} && (1.31)
\end{aligned}$$

where the first term is the KL-divergence between the approximate and the true posterior and $\mathcal{F}_{\mathbf{x}}$ is usually referred to as *evidence lower bound* (ELBO) of the

2 Probabilistic PCA is a notable exception to this

datapoint $\mathbf{x}$. As the KL divergence is non-negative, the ELBO is a lower bound to the marginal likelihood. It is usually written as

$$\mathcal{F}_{\mathbf{x}} = \mathbb{E}_{q(\mathbf{z} \mid \mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}) - \mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})), \tag{1.32}$$

where the first term in Equation (1.32), the (negative) log reconstruction error, measures how well a sample $\mathbf{x}$ can be reconstructed by the model from its latent representation produced by the approximate posterior. The second term, the KL divergence between the approximate posterior and the prior, acts as a regulariser and limits the information passing through the latent space and ensures that the model can also generate the data rather than just reconstruct it. Note that $\mathcal{F}_{\mathbf{x}}$ denotes the ELBO for a specific datapoint $\mathbf{x}$ and the variational distribution $q(\mathbf{z} \mid \mathbf{x})$ is also particular to this datapoint. In general, we therefore introduce one approximate variational distribution per datapoint.

To evaluate the ELBO, we have to specify the prior, likelihood, and approximate posterior. In general, by choosing a Gaussian approximate posterior as well as a (standard Normal) Gaussian prior, the KL term can be computed in closed form:

$$\left.\begin{aligned} q(\mathbf{z} \mid \mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}, \Sigma) \\ p(\mathbf{z}) &= \mathcal{N}(0, \mathbb{1}) \end{aligned}\right\} \curvearrowright \mathrm{KL}(q \parallel p) = \tfrac{1}{2}\left(\operatorname{tr}\Sigma + \boldsymbol{\mu}^{\intercal}\boldsymbol{\mu} - D - \log\det\Sigma\right) \tag{1.33}$$

$$\left.\begin{aligned} q(\mathbf{z} \mid \mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \\ p(\mathbf{z}) &= \mathcal{N}(0, \mathbb{1}) \end{aligned}\right\} \curvearrowright \mathrm{KL}(q \parallel p) = \tfrac{1}{2}\sum_{d=1}^{D}\left(\sigma_d^2 + \mu_d^2 - \log\sigma_d^2 - 1\right), \tag{1.34}$$

where we have assumed a full covariance approximate posterior in Equation (1.33) and a diagonal approximate posterior in Equation (1.34). Note that so far the means and (co)variances are still datapoint dependent; that is, we are in a variational setting but do not employ amortisation (yet).

Depending on the likelihood model, the log reconstruction error in Equation (1.32) may be tractable (for Probabilistic PCA) or intractable (for most other models). In the latter case, one usually reverts to estimating the expectation by a number of Monte Carlo (MC) samples. In that case we approximate

$$\mathbb{E}_{q(\mathbf{z} \mid \mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}) \approx \tfrac{1}{M}\sum_{m=1}^{M} \log p(\mathbf{x} \mid \mathbf{z}_m) \qquad \mathbf{z}_m \sim q(\mathbf{z} \mid \mathbf{x}), \tag{1.35}$$

where M is the number of MC samples used. Often, one chooses $M = 1$, that is, we only use a single sample to estimate the expectation.

When the prior or approximate posterior is non-Gaussian, the KL term is also intractable and usually estimated from samples as well. Interestingly, even

in cases where the KL divergence can be computed in closed form, it may be beneficial to sample it instead (Roeder et al., 2017). This is because the two terms – when both are sampled – may be correlated or anti-correlated in such a way as to yield smaller gradient variances than when only one of the two terms is sampled. According to Roeder et al. (2017) this effect may be particularly pronounced close to an optimum.

1.6 VARIATIONAL AUTOENCODERS

Variational Autoencoders (VAEs) (P. and Welling, 2014; Rezende et al., 2014) are powerful and widely used probabilistic deep generative models. They are also continuous latent variable models such that the discussion in Section 1.5 also applies to them. Their main innovation is the usage of amortised variational inference and stochastic optimisation with the help of the reparameterisation trick, as we explain in the following. Amortised inference refers to the fact that we do not optimise per-datapoint variational parameters, but instead train a recognition model that outputs the variational parameters in a single forward pass; the cost of inference is amortised among all the datapoints. Combined with deep neural networks for the likelihood function and the recognition model, this enables VAEs to scale to very large datasets and allows for fast test-time inference.

Due to the superficial similarities to other autoencoder models such as denoising autoencoders (Vincent et al., 2008), the recognition model or approximate posterior is also referred to as the *encoder* whereas the likelihood is referred to as the *decoder*. Intuitively speaking, this is because the encoder *encodes* datapoints $\mathbf{x}$ into latent codes $\mathbf{z}$ whereas the decoder *decodes* latent codes $\mathbf{z}$ back into reconstructed datapoints $\mathbf{x}$.

encoder

The encoder network parameterises the parameters of the approximate posterior $q_\varphi(\mathbf{z} \mid \mathbf{x})$ through neural network models. In the simplest case of a diagonal Gaussian approximate posterior (P. and Welling, 2014), this corresponds to:

$$\begin{aligned} q_\varphi(\mathbf{z} \mid \mathbf{x}) &= \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\varphi(\mathbf{x}), \operatorname{diag} \boldsymbol{\sigma}_\varphi(\mathbf{x})\right) \\ \boldsymbol{\mu}_\varphi(\mathbf{x}) &= \mathbf{f}_\varphi(\mathbf{x}) \\ \operatorname{diag} \boldsymbol{\sigma}^2_\varphi(\mathbf{x}) &= \exp\left(\mathbf{g}_\varphi(\mathbf{x})\right) \end{aligned} \tag{1.36}$$

where the mean and log-standard deviations[3] parameters of the Gaussian are themselves functions $\mathbf{f}_\varphi$ and $\mathbf{g}_\varphi$ of the inputs. In VAEs, these functions are parameterised by deep neural networks and φ refers to its parameters. Usually,

3 We usually parameterise the log-standard deviation instead of the variances to transform the constrained optimisation problem (positive variances) into an unconstrained optimisation problem (real valued log-standard deviations).

these networks are shared up to the last layer, which then splits into two heads for the mean and log-variances, respectively.

In VAEs, all parameters of the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$, such as means, log-scales, or mixture weights, are again determined through a neural network referred to as decoder. These are usually shared up to the last layer, as well. Note that conceptually, this is no different from Probabilistic PCA, where the variance of the likelihood is fixed and the means are given by a linear decoding of the latent codes $\mathbf{z}$. In VAEs, these linear mappings are replaced by non-linear functions and applied to other parameters of the likelihood as well. As discussed above, typical likelihood functions include the Gaussian and the Bernoulli distribution but also more complicated auto-regressive structures or the discretised mixture of logistics likelihood (Salimans et al., 2017).

decoder

1.6.1 *Training VAEs*

In order to train a VAE model, we would like to maximise the log marginal likelihood of the training data. Because of its intractability, we optimise the evidence lower bound in Equation (1.32) instead, where we replace the per-datapoint approximate posterior $q(\mathbf{z} \mid \mathbf{x})$ with the amortised distribution $q_\varphi(\mathbf{z} \mid \mathbf{x})$. P. and Welling (2014) propose to jointly optimise the objective with respect to all the model and variational parameters. Alternatively, other iterative or EM style algorithms only update some of the parameters at a time, such as the influential wake-sleep algorithm (Hinton et al., 1995). As we discussed earlier, at least the log reconstruction part of the ELBO objective needs to be evaluated using MC samples from the approximate posterior $q_\varphi(\mathbf{z} \mid \mathbf{x})$. Thus, during training, we need to sample from q_φ, and typically use a single sample to estimate the expectation in Equation (1.32). However, the sample or samples depend on the encoder parameters φ, such that it appears like we must differentiate through (discrete) samples in order to backpropagate changes into these parameters. Note that this problem does not arise when evaluating gradients for the decoder, because we never sample from the model but evaluate the likelihood at training inputs, which is continuous.

In principle, we could use REINFORCE (Williams, 1992) to estimate the gradients with respect to the variational encoder parameters; however, these gradient estimates have a very high variance which often precludes efficient learning. Luckily, many distributions can be *reparameterised* into a deterministic and a stochastic component, where only the deterministic component depends on the variational parameters. Through this so-called *reparameterisation trick* (P. and Welling, 2014) we effectively do not have to differentiate through the sampling procedure. The resulting gradients are also unbiased (similarly to REINFORCE

reparameterisation trick

gradients) but show much lower variance. For distributions that are not easily reparameterisable, many gradient estimators have been developed over the years, for example for discrete latent variables (Maddison et al., 2017; Tucker et al., 2017) or implicit reparameterisation (Figurnov et al., 2018). Mohamed et al. (2019) provide an excellent recent review of Monte Carlo gradient estimation in machine learning.

1.6.2 *More flexible approximate posteriors*

Since the inception of VAEs many extensions and improvements have been reported, some of which we already mentioned above, such as richer likelihood models or advances in gradient estimation. Another branch of the literature investigates richer and more flexible approximate posterior distributions instead of the simple diagonal Gaussian encoder in Equation (1.36). Examples include but are not limited to MCMC based approaches (Salimans et al., 2015), semi-amortised approaches (Kim et al., 2018), or flow-based methods that we detail now.

normalising flows

One important family of flexible distributions, which we also employ in this thesis, are *normalising flows* presented by Tabak and Vanden-Eijnden (2010) and Tabak and Turner (2013) and introduced in machine learning by Rezende and Mohamed (2015). Normalising flows are continuously differentiable bijections that map densities to densities while allowing us to efficiently compute the change in volume due to the transformation. For this, they rely on the change of variables formula:

Theorem 1.1 (Change of variables formula)
Let $\mathbf{z} \in \mathbb{R}^d$ be a random variable with density $p(\mathbf{z})$ and $f : \mathbb{R}^d \to \mathbb{R}^d$ a smooth invertible mapping. We can then use f to transform $\mathbf{z} \sim p(\mathbf{z})$ to a new random variable $\mathbf{y} = f(\mathbf{z})$ which has the following density:

$$p(\mathbf{y}) = p(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{z}} \right| = p(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1} . \tag{1.37}$$

Thus, to evaluate the change of density we need to be able to efficiently compute the determinant of the Jacobian matrix. This is possible for the different flows that have been proposed in the literature, such as planar and radial flows (Rezende and Mohamed, 2015), inverse autoregressive flows (IAF) (Kingma et al., 2016), real non-volume preserving flows RealNVP (Dinh et al., 2017), or masked autoregressive flows (MAF) (Papamakarios et al., 2017).

Typically, the encoder outputs a Gaussian distribution $q_0(\mathbf{z}_0 \mid \mathbf{x})$, which is then transformed through a series of flows,

$$\mathbf{z}_K = f_K \circ \cdots \circ f_1(\mathbf{z}_0) \qquad \mathbf{z}_0 \sim q_0(\mathbf{z}_0 \mid \mathbf{x}) \tag{1.38}$$

$$q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0 \mid \mathbf{x}) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1}, \tag{1.39}$$

and both the encoder as well as the flow parameters are amortised.

1.6.3 *Model evaluation: the importance weighted bound*

While VAEs are typically trained using the ELBO, they are numerically compared on tighter estimates of the log marginal likelihood. The standard procedure for this is to use a tighter bound introduced with importance weighted auto-encoders (IWAE) (Burda et al., 2016) that we briefly explain in the following. Burda et al. (2016) note that the ELBO in Equation (1.32) can also be directly approximated with several samples:

importance sampling

$$\mathcal{F}_{\mathbf{x}} = \mathbb{E}_{q(\mathbf{z} \mid \mathbf{x})} \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})} \right] \tag{1.40}$$

$$\approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k)}{q(\mathbf{z}_k \mid \mathbf{x})} \qquad \mathbf{z}_k \overset{\text{i.i.d}}{\sim} q(\mathbf{z} \mid \mathbf{x}) \tag{1.41}$$

$$=: \mathcal{F}_{\mathbf{x}}^{(k)} \tag{1.42}$$

where the ratio $w_k = \frac{p(\mathbf{x}, \mathbf{z}_k)}{q(\mathbf{z}_k \mid \mathbf{x})}$ can be interpreted as unnormalised importance weights of the joint distribution. In usual VAE training only a single sample $\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{x})$ is used to evaluate the objective. Burda et al. (2016) go on to show that this bound can be made tighter and the true marginal likelihood can be better estimated using more samples, see Theorem 1.2.

Theorem 1.2 (Theorem 1 in (Burda et al., 2016))
For all k, the lower bounds satisfy

$$\log p(\mathbf{x}) \geq \mathcal{F}_{\mathbf{x}}^{(k+1)} \geq \mathcal{F}_{\mathbf{x}}^{(k)} . \tag{1.43}$$

Moreover, if $\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})}$ is bounded, then $\mathcal{F}_{\mathbf{x}}^{(k)}$ approaches $\log p(\mathbf{x})$ as k goes to infinity.

Recent results by Rainforth et al. (2018) show that using these tighter bounds is not beneficial for training because the gradients with respect to the encoder parameters decay towards zero as K gets larger. In particular, they decay faster than the variance of the estimator decreases (which would help training in principle).

Most papers report a log marginal likelihood estimate based on 1000 to 4000 importance samples. Often, the results are simply quoted as $\log p$.

1.6.4 *Rewriting the ELBO*

While Equation (1.32) is the standard form of the ELBO that is also used when implementing it algorithmically, it can be re-written in several ways that highlight different properties and allow for different analyses.

variational free energy

Especially within the statistical physics community, $\mathcal{F}$ is also referred to as the *(variational) free energy*. This is because it can be re-written as

$$\mathcal{F} = \int q(\mathbf{z} \mid \mathbf{x}) \log p(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z} + \underbrace{\int q(\mathbf{z} \mid \mathbf{x}) \log \frac{1}{q(\mathbf{z} \mid \mathbf{x})} \, \mathrm{d}\mathbf{z}}_{= \mathcal{H}(q(\mathbf{z} \mid \mathbf{x}))}, \tag{1.44}$$

where the first term can be interpreted as a (negative) energy and the second term is the entropy, refer to MacKay (2003) for a detailed discussion. Such a sum of energy and entropy is referred to as the *free energy* of a system in the physics literature. To be more precise, $-\log p(\mathbf{y})$ would be referred to as the Helmholtz free energy, whereas $-\mathcal{F} = -\log p(\mathbf{y}) + \mathrm{KL}$ would be referred to as Gibbs free energy, see, e.g., Yedidia et al. (2005) for a review of free-energy approximations in graphical models.

Equation (1.32) denotes the ELBO for a single data point $\mathbf{x}$; we obtain the ELBO for the dataset by averaging it over the empirical distribution of the dataset $p_{\text{Data}}(\mathbf{x})$

$$\mathcal{F} = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})} \mathcal{F}_{\mathbf{x}} = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})} \left[\mathbb{E}_{q_\varphi(\mathbf{z} \mid \mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z}) - \mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) \right]. \tag{1.45}$$

aggregate posterior

Hoffman and Johnson (2016) propose to rewrite the KL term in Equation (1.45) in terms of the *aggregate approximate posterior*,

$$q(\mathbf{z}) = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})} q(\mathbf{z} \mid \mathbf{x}), \tag{1.46}$$

which can be interpreted as the average encoder distribution, or in other words, it is the aggregate distribution of all embedded training examples. The KL term is given by

$$\mathbb{E}_{p_{\text{Data}}(\mathbf{x})} \mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) = \mathrm{KL}(q(\mathbf{z}) \parallel p(\mathbf{z})) + \underbrace{\int q(\mathbf{z}, \mathbf{x}) \log \frac{q(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}) p_{\text{Data}}(\mathbf{x})}}_{\text{mutual information } \mathbb{I}(\mathbf{z};\mathbf{x})}, \tag{1.47}$$

and the full average ELBO can be written as

$$\mathcal{F} = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})} \left[\mathbb{E}_{q_\varphi(\mathbf{z} \mid \mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z}) \right] - \mathrm{KL}(q(\mathbf{z}) \parallel p(\mathbf{z})) - \mathbb{I}(\mathbf{z}; \mathbf{x}) \,, \tag{1.48}$$

where last term denotes the *mutual information* between the latent vectors and the training examples. It is given by the KL between the joint and the marginal distributions, $\mathrm{KL}(q(\mathbf{z}, \mathbf{x}) \parallel q(\mathbf{z}) p_{\text{Data}}(\mathbf{x}))$. It is symmetric, positive, and zero if and only if the joint distribution factorises, that is, when $\mathbf{z}$ holds no information about $\mathbf{x}$. Maximisation of Equation (1.48) implies minimisation of the mutual information.

mutual information

In this form, Equation (1.48), we can directly compare the ELBO objective to the relaxed objective function of the more recently introduced *Wasserstein autoencoders* (WAE) (Tolstikhin et al., 2018). By choosing the log likelihood as cost function and the KL as divergence measure in Equation (4) of Tolstikhin et al. (2018), we find that it is identical to Equation (1.48) except for mutual information term. As Hoffman and Johnson (2016) discuss, this mutual information term can be seen as an additional regulariser that tries to make $\mathbf{x}$ and $\mathbf{z}$ independent of each other. This goal is counter to our main objective in unsupervised learning and generative modelling of learning meaningful latent representations.

One particularly egregious example of this over-regularisation, the so-called *posterior collapse*, has been reported by Bowman et al. (2016) and discussed by Chen et al. (2017) and Zhao et al. (2019) among others: For flexible likelihood models – especially when they are autoregressive – the approximate posterior $q_\varphi(\mathbf{z} \mid \mathbf{x})$ can collapse to the prior $p(\mathbf{z})$, such that $\mathbf{x}$ and $\mathbf{z}$ become completely independent.

posterior collapse

This collapse can also occur for only some dimensions of the latent space, which lead to the distinction between active (i.e. coding) and inactive (non-coding) units (Burda et al., 2016). This behaviour is to be expected from VI and is caused by the same mechanism that typically causes it to underfit (see (Turner and Sahani, 2011, Problem 2)).

Part I

PROBABILISTIC INFERENCE IN GAUSSIAN PROCESSES

2

INTRODUCTION TO SPARSE GAUSSIAN PROCESS APPROXIMATIONS

This chapter introduces sparse approximation methods for Gaussian process models. These form the basis of recent advances in probabilistic inference with Gaussian processes, which are presented in Chapters 3 and 4.

2.1 SUBSET OF DATA

Subset of data (SoD) is a simple method to reduce complexity by discarding data, retaining only M datapoints. SoD with M randomly selected datapoints reduces computational complexity of inference in Gaussian processes to $\mathcal{O}(M^3)$ but may not yield competitive results, and uncertainties may be modeled poorly (Quiñonero-Candela and Rasmussen, 2005).

Some datapoints are usually more informative than others. Greedy selection methods have been developed to select data based on different information theoretic criteria (refer to, for example, Seeger et al. (2003) and Lawrence et al. (2003)). This typically decreases the computational complexity to $\mathcal{O}(NM^2)$. SoD is often used in practice due to its conceptual simplicity and low computational overhead, even though more elaborate methods yield better results.

2.2 OVERVIEW OF SPARSE APPROXIMATIONS

In this section we focus on methods that approximate the posterior using the function values at M *inducing inputs* (aka pseudo-points). These methods comprise the Subset of Regressor (SoR) (Wahba, 1990; Smola and Bartlett, 2001), the Deterministic Training Conditional (DTC) (Seeger et al., 2003), the Fully Independent Training Conditional (FITC) (Snelson and Ghahramani, 2006), and a variational approximation that we refer to as Variational Free Energy (VFE) (Titsias, 2009a).

The improved performance in all these methods relies on a low rank approximation $Q_{\mathbf{ff}}$ to the full covariance matrix $K_{\mathbf{ff}}$, that is parameterised in terms of the M inducing variables. Figure 2.1 illustrates the placement of the input location of these inducing variables for VFE, a sparse method in which the locations of the inducing inputs are learned, see Section 2.4. The intuition behind these sparse approximations is, that the eigenvalue spectrum of the covariance matrix often

decays rapidly and can be approximated by the most dominant components (Snelson, 2007). As we have noted in Section 1.4.4, an exact eigendecomposition takes $\mathcal{O}(N^3)$ computations and is, therefore, not feasible.

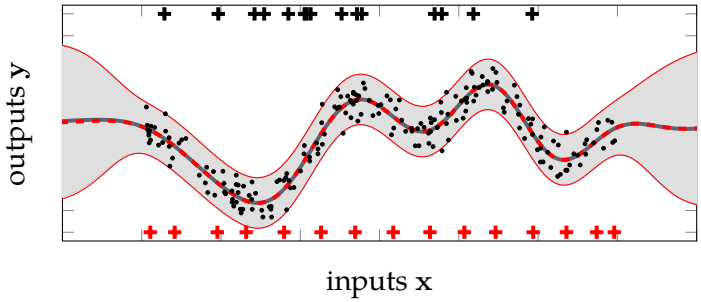


Figure 2.1: Posterior predictive distribution for VFE using 15 inducing variables whose location in input space has been optimised. Black crosses **+**: initial locations before optimisation, red crosses **+**: optimised location.

Many of the sparse methods in this section have been reviewed and re-derived in a unified framework by Quiñonero-Candela and Rasmussen (2005). We follow their exposition.

inducing variables

First, we extend the prior over $\mathbf{f}, \mathbf{f}^*$ by a set of latent *inducing variables* that are located at *inducing inputs* $Z = (\mathbf{z}_1, \ldots, \mathbf{z}_M)^\intercal$ and have *inducing ouputs* $\mathbf{u} = (u_1, \ldots, u_M)^\intercal$. Z lives in the same space as the training inputs X, whereas, for now, $\mathbf{u}$ lives in the same space as $\mathbf{f}$. Because of the consistency property of GPs, the original prior over $\mathbf{f}$ and $\mathbf{f}^*$ can be recovered by marginalising over $\mathbf{u}$:

$$\begin{aligned} p(\mathbf{f}, \mathbf{f}^*) &= \int p(\mathbf{f}, \mathbf{f}^*, \mathbf{u}) \, \mathrm{d}\mathbf{u} = \int p(\mathbf{f}, \mathbf{f}^* \mid \mathbf{u}) p(\mathbf{u}) \, \mathrm{d}\mathbf{u} \\ p(\mathbf{u}) &= \mathcal{N}(0, K_{\mathbf{uu}}), \end{aligned} \tag{2.1}$$

where $[K_{\mathbf{uu}}]_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$ is the covariance matrix of the inducing inputs. To make progress towards faster inference, we introduce a first approximation and assume that $\mathbf{f}$ and $\mathbf{f}^*$ are *conditionally independent* given $\mathbf{u}$; that is, we assume $\mathbf{u}$ contains all the dependencies between training and test data,

$$p(\mathbf{f}, \mathbf{f}^*) \approx q(\mathbf{f}, \mathbf{f}^*) = \int q(\mathbf{f} \mid \mathbf{u}) q(\mathbf{f}^* \mid \mathbf{u}) p(\mathbf{u}) \, \mathrm{d}\mathbf{u}. \tag{2.2}$$

In general, this assumption does not hold and is made as a form of approximation. Figure 2.2 shows the factor graph representation of the full GP and the conditional independence approximation with inducing variables. Such a concentration of the inference on the M inducing variables is conceptually similar to an information bottleneck in variational autoencoders (P. and Welling, 2014). Most of the following sparse approximations in this section make further assumptions on the form of

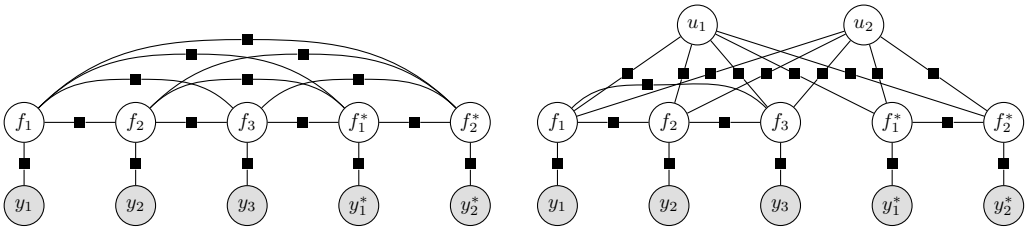


Figure 2.2: Factor graphs for the full GP *(left)* and conditional independence approximation after introduction of the inducing variables *(right)*.

$q(\mathbf{f} \mid \mathbf{u})$ and $q(\mathbf{f}^* \mid \mathbf{u})$. Their idea is to approximate the prior of the full GP further, such that exact inference in the approximate model is computationally tractable.

For further reference, the exact conditional distributions are:

$$p(\mathbf{f} \mid \mathbf{u}) = \mathcal{N}\left(\mathbf{f}; K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}\mathbf{u}, K_{\mathbf{ff}} - Q_{\mathbf{ff}}\right) \tag{2.3a}$$

$$p(\mathbf{f}^* \mid \mathbf{u}) = \mathcal{N}\left(\mathbf{f}; K_{*\mathbf{u}}K_{\mathbf{uu}}^{-1}\mathbf{u}, K_{**} - Q_{**}\right), \tag{2.3b}$$

where $Q_{\mathbf{a},\mathbf{b}} = K_{\mathbf{au}}K_{\mathbf{uu}}^{-1}K_{\mathbf{ub}}$ is the so-called *Nyström approximation* of the covariance matrix $K_{\mathbf{ab}}$ between any two vectors $\mathbf{a}$ and $\mathbf{b}$ and $*$ denotes evaluation at test inputs $\mathbf{x}^*$; i.e., $K_{**} = k(\mathbf{x}^*, \mathbf{x}^*)$. The exact conditionals (Equation (2.3)) are very similar to those in Equation (1.13) as they are both conditionals of normally distributed variables. Note that the approximate prior (Equation (2.2)) is independent of the inducing outputs $\mathbf{u}$ but does depend on the inducing inputs Z that appear in the covariance matrix $K_{\mathbf{uu}}$.

Nyström approximation

The Nyström approximation yields a low-rank approximation to the eigendecomposition of $K_{\mathbf{ab}}$ (refer to Rasmussen and Williams (2006, Sec. 8.1)). This is illustrated in Figure 2.3 for the training covariance matrix $K_{\mathbf{ff}}$, which is utilised in all of the approximations in this section. Using the matrix inversion lemma (refer to Petersen and Pedersen (2012, Sec. 3.2)), the inverse of the approximate covariance matrix $Q_{\mathbf{ff}}$ can be computed efficiently and the computational cost of these methods is dominated by matrix multiplications of the form $(Q_{\mathbf{ff}} + \sigma_n^2)^{-1}K_{\mathbf{uf}}$, which takes $\mathcal{O}(NM^2)$ computations and $\mathcal{O}(NM)$ memory.

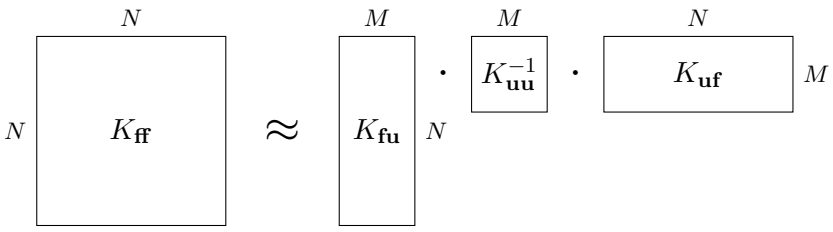


Figure 2.3: Schematic illustration of the Nyström low rank approximation of the covariance matrix $K_{\mathbf{ff}} \approx Q_{\mathbf{ff}} = K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}K_{\mathbf{uf}}$. To achieve a computational speedup we have to choose $M \ll N$.

Here, we treat the inducing variable outputs $\mathbf{u}$ as latent function observations, or as living in the same space as $\mathbf{f}$. In principle, $\mathbf{u}$ can be any quantity that is informative about the GP, such as derivatives, integrals, or even mixtures thereof. Sparse approximations in such a setting are referred to as *inter-domain GPs* (Lázaro-Gredilla and Figueiras-Vidal, 2009) or *inducing kernels* (Álvarez et al., 2010) and play an important role in making convolutional GPs (van der Wilk et al., 2017) as well as invariant GPs (see Chapter 4) computationally tractable.

inter-domain GPs

2.2.1 *Nyström method*

Simply approximating $K_{\mathbf{ff}}$ by $Q_{\mathbf{ff}} = K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}K_{\mathbf{uf}}$ in Equation (1.13) is also referred to as the Nyström method and was introduced as a GP approximation by Williams and Seeger (2001). They choose $\mathbf{u}$ as a subset of the latent variables $\mathbf{f}$. This approximation is only performed on the training covariance matrix; it can be interpreted as a model with *inconsistent* joint prior (Quiñonero-Candela and Rasmussen, 2005):

$$q_{\text{Nyst}}(\mathbf{f}, \mathbf{f}^* \mid X, X^*) = \mathcal{N}\left(0, \begin{bmatrix} Q_{\mathbf{ff}} & K_{\mathbf{f}*} \\ K_{*\mathbf{f}} & K_{**} \end{bmatrix}\right) \tag{2.4}$$

As only the matrix $K_{\mathbf{ff}}$ is replaced and not the entire covariance function, this approximation can lead to nonsensical results such as negative predictive variances (Rasmussen and Williams, 2006). Yet, for large number of inducing variables M, the approximation can yield reasonable results that are, however, superseded by the other approximations we present next.

2.2.2 *Deterministic Training Conditional*

The Deterministic Training Conditional (DTC) was introduced by Seeger et al. (2003) under the original name of Projected Latent Variables (PLV) building on the ideas of Csató and Opper (2002). Originally designed as a likelihood approximation, it can be re-phrased as the following prior approximation,

$$q_{\text{DTC}}(\mathbf{f} \mid \mathbf{u}) = \mathcal{N}\left(K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}\mathbf{u}, 0\right) \tag{2.5a}$$

$$q_{\text{DTC}}(\mathbf{f}^* \mid \mathbf{u}) = p(\mathbf{f}^* \mid \mathbf{u}) \tag{2.5b}$$

$$q_{\text{DTC}}(\mathbf{f}, \mathbf{f}^* \mid X, X^*) = \mathcal{N}\left(0, \begin{bmatrix} Q_{\mathbf{ff}} & Q_{\mathbf{f}*} \\ Q_{*\mathbf{f}} & K_{**} \end{bmatrix}\right). \tag{2.5c}$$

Here, the values for the latent function values $\mathbf{f}$ are fully determined by the inducing variables. This prior approximation leads to the following predictive distribution:

$$p_{\mathrm{DTC}}(\mathbf{f}^* \mid X, \mathbf{y}, X^*) = \mathcal{N}(\mathbf{f}^* \mid \boldsymbol{\mu}_{\mathrm{DTC}}, \Sigma_{\mathrm{DTC}}) \tag{2.6a}$$

$$\boldsymbol{\mu}_{\mathrm{DTC}} = \sigma^{-2} K_{*\mathbf{u}} (K_{\mathbf{uu}} + \sigma^{-2} K_{\mathbf{fu}} K_{\mathbf{fu}})^{-1} K_{\mathbf{uf}} \mathbf{y} \tag{2.6b}$$

$$\Sigma_{\mathrm{DTC}} = K_{**} - Q_{**} + K_{*\mathbf{u}} (K_{\mathbf{uu}} + \sigma^{-2} K_{\mathbf{uf}} K_{\mathbf{fu}})^{-1} K_{\mathbf{u}*} \tag{2.6c}$$

and this marginal likelihood:

$$\log p_{\mathrm{DTC}}(\mathbf{y} \mid X, \boldsymbol{\theta}) = \log \mathcal{N}(\mathbf{y} \mid 0, Q_{\mathbf{ff}} + \sigma_n^2 I) \tag{2.7}$$

It should be noted that Equation (2.5) does not correspond to a consistent GP, as the covariance function for the training and test latent variables is different.

Similar to the previously discussed methods, the inducing inputs Z (also referred to as *active set*) that parameterise the covariance matrix $K_{\mathbf{uu}}$ are chosen from the data inputs X. Csató and Opper (2002) and Seeger et al. (2003) propose two different criteria for their selection (also see Rasmussen and Williams (2006, Chapter 8.3)). The hyperparameters $\boldsymbol{\theta}$ are learned by maximising the marginal likelihood Equation (2.7). These two separate procedures – selection of the active set and hyperparameter-optimisation – are performed iteratively.

2.3 THE FULLY INDEPENDENT TRAINING CONDITIONAL

Snelson and Ghahramani (2006) introduced the Fully Independent Training Conditional (FITC) under the name Sparse Pseudo-inputs GP (SPGP). Initially derived as a likelihood approximation, it can again be re-phrased as a prior approximation (Quiñonero-Candela and Rasmussen, 2005), whereby we assume that the training latent function values $\mathbf{f}$ are conditionally independent given the inducing variables $\mathbf{u}$, see Figure 2.4 and Equation (2.8). An extension of FITC called FIC also assumes that the latent test values $\mathbf{f}^*$ are conditionally independent, see the blue terms in Equation (2.8).

$$q_{\mathrm{FITC}}(\mathbf{f} \mid \mathbf{u}) = \prod_{j=1}^{N} p(f_j \mid \mathbf{x}_j, \mathbf{u}) \tag{2.8a}$$

$$q_{\mathrm{FITC}}(\mathbf{f}^* \mid \mathbf{u}) = p(\mathbf{f}^* \mid \mathbf{u}) \qquad q_{\mathrm{FIC}}(\mathbf{f}^* \mid \mathbf{u}) = \prod_{j=1}^{N^*} p(f_j^* \mid \mathbf{x}_j^*, \mathbf{u}) \tag{2.8b}$$

$$q_{\mathrm{FITC}}(\mathbf{f}, \mathbf{f}^* \mid X, X^*) = \mathcal{N}\left(0, \begin{bmatrix} Q_{\mathbf{ff}} + \operatorname{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}) & Q_{\mathbf{f}*} \\ Q_{*\mathbf{f}} & K_{**} \end{bmatrix}\right) \tag{2.8c}$$

Observe in Equation (2.8) that FITC uses the Nyström approximation $Q_{\mathbf{ff}}$ for the off-diagonal elements of the covariance matrix but is exact (i.e. uses the elements of $K_{\mathbf{ff}}$) on the diagonal.

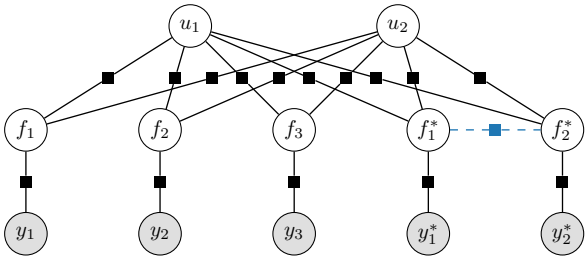


Figure 2.4: Graphical model for FITC and FIC. Compared to Figure 2.2 *(right)* the links between the individual latent function values f_i have been removed to reflect the conditional independence assumption in Equation (2.8). For FIC, the factors connecting the test latent variables f_i^* are dropped as well.

Performing exact Bayesian inference with the FITC prior leads to the following predictive distribution:

$$p_{\mathrm{FITC}}(\mathbf{f}^* \mid \mathbf{y}, X, X^*) = \mathcal{N}\left(\boldsymbol{\mu}_{\mathrm{FITC}}, \Sigma_{\mathrm{FITC}}\right) \tag{2.9a}$$

$$\boldsymbol{\mu}_{\mathrm{FITC}} = K_{*\mathbf{u}}(K_{\mathbf{uu}} + K_{\mathbf{fu}}G^{-1}K_{\mathbf{fu}})^{-1}K_{\mathbf{uf}}G^{-1}\mathbf{y} \tag{2.9b}$$

$$\Sigma_{\mathrm{FITC}} = K_{**} - Q_{**} + K_{*\mathbf{u}}(K_{\mathbf{uu}} + K_{\mathbf{uf}}G^{-1}K_{\mathbf{fu}})^{-1}K_{\mathbf{u}*} \tag{2.9c}$$

where $G = \mathrm{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}) + \sigma_n^2 I$. This term can be interpreted as an input dependent or *heteroscedastic* noise term (Snelson and Ghahramani, 2006), that we discuss in detail in Chapter 3. The FITC negative marginal likelihood is given by:

$$\mathcal{F}_{\mathrm{FITC}} = -\log p(\mathbf{y} \mid X) = \frac{N}{2}\log(2\pi) + \frac{1}{2}\log|Q_{\mathbf{ff}} + G| + \frac{1}{2}\mathbf{y}^{\mathsf{T}}(Q_{\mathbf{ff}} + G)^{-1}\mathbf{y} \tag{2.10}$$

Learning of FITC models can again be done by type II maximum likelihood learning, that is, by optimising the marginal likelihood Equation (2.10) with respect to the hyperparamters $\boldsymbol{\theta}$. In addition, Snelson and Ghahramani (2006) also use it to optimise the inducing variable inputs Z. This can, in principle, lead to overfitting as we optimise quite a number of variables. We discuss this issue further in Chapter 3.

Moreover, only FIC corresponds to a consistent GP because FITC, like DTC, uses a different covariance function for training and test latent variables. However, in practice, FITC is much more widely used, since an approximation of the test covariances is usually not necessary from a computational point of view.

Here, we have introduced FITC as an approximate model, in which exact inference can be performed efficiently. FITC can also be derived by approximating the exact GP posterior Equation (1.13) using Expectation Propagation (EP, cf. Minka (2001)). Refer to Csató and Opper (2001, 2002) for the original publications

on using EP in this context and Qi et al. (2010) for applying EP in models with generalised (non-Gaussian) likelihoods. Bui et al. (2016) unify and re-derive FITC from a generalised Power Expecation Propagation treatment that also includes the Variational Free Energy approximation (cf. Section 2.4).

We provide a detailed and comparative discussion between VFE and FITC in Chapter 3

2.4 THE VARIATIONAL FREE ENERGY METHOD

An alternative and in some ways more principled approach that also extends to non-conjugate GP models is to apply variational inference. In the context of GPs, variational inference was introduced by Titsias (2009a) (refer to Titsias (2009b) for an extended technical report). We refer to this approximation as Variational Free Energy (VFE) approximation. While the previously presented methods approximate the model and then perform exact inference, VFE performs approximate inference in the original model.

Like the previous sparse approximations methods, VFE first augments the model by inducing variables at input locations Z and with output values $\mathbf{u}$. The "intractable" distribution we wish to approximate is the posterior probability $p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})$, which can be factorised as

$$p(\mathbf{f}, \mathbf{u} \mid \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u}, \mathbf{y}) p(\mathbf{u} \mid \mathbf{y}). \tag{2.11}$$

We approximate it by a variational distribution $q(\mathbf{f}, \mathbf{u})$ that we factorise as

$$q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u}) \phi(\mathbf{u}) \,, \tag{2.12}$$

where $p(\mathbf{f} \mid \mathbf{u})$ is the exact conditional (Equation (2.3)) and $\phi(\mathbf{u})$ is a free form distribution. In Equation (2.12) we have made two approximations; first, we drop the dependency on $\mathbf{y}$ in the conditional for $\mathbf{f}$, similar to the independence assumptions above. Second, we replace $p(\mathbf{u} \mid \mathbf{y})$ by the free form variational distribution $\phi(\mathbf{u})$, which we choose by minimising

$$\mathrm{KL}(q(\mathbf{f}, \mathbf{u}) \parallel p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})) = \int q(\mathbf{f}, \mathbf{u}) \log \frac{q(\mathbf{f}, \mathbf{u})}{p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})} \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{u}, \tag{2.13}$$

that is, by demanding that $q(\mathbf{f}, \mathbf{u})$ is as close as possible to $p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})$. Because Equation (2.13) contains the intractable posterior, we cannot directly compute it and consider the evidence lower bound (ELBO) to the log marginal likelihood

instead, similar to how we derived the ELBO for continuous latent variable models in Section 1.5:

$$\begin{aligned}\log p(\mathbf{y}) &\overset{(a)}{=} \log \frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u})}{p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})} \\ &\overset{(b)}{=} \int q(\mathbf{f}, \mathbf{u}) \log \left(\frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u})}{p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})} \frac{q(\mathbf{f}, \mathbf{u})}{q(\mathbf{f}, \mathbf{u})} \right) \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u} \\ &= \underbrace{\int q(\mathbf{f}, \mathbf{u}) \log \frac{q(\mathbf{f}, \mathbf{u})}{p(\mathbf{f}, \mathbf{u} \mid \mathbf{y})} \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u}}_{\text{KL Equation (2.13)}} + \underbrace{\int q(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u})}{q(\mathbf{f}, \mathbf{u})} \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u}}_{=: \mathcal{F}} \geq \mathcal{F}\end{aligned} \tag{2.14}$$

where equality (a) is the formula for conditionals and equality (b) is taking expectation w.r.t. $q(\mathbf{f}, \mathbf{u})$ because the left hand side does not depend on $\mathbf{f}$ and $\mathbf{u}$, and we have multiplied the argument of the log by one, effectively. The result consists of two terms, the KL divergence Equation (2.13) and $\mathcal{F}$ again denotes the ELBO to the true marginal likelihood, which we interchangeably refer to as variational free energy.

As the KL is always non-negative, the variational free energy $\mathcal{F}$ is a lower bound to the marginal likelihood $\log p(\mathbf{y})$. Thus, maximising $\mathcal{F}$ w.r.t. the variational distribution $q(\mathbf{f}, \mathbf{u})$ is equivalent to minimising the KL. In other words, to make the bound as tight as possible, we choose the best approximate posterior distribution $q(\mathbf{f}, \mathbf{u})$:

$$\log p(\mathbf{y}) \geq \mathcal{F}_{\text{opt}} = \max_{q(\mathbf{f},\mathbf{u})} \mathcal{F}[q(\mathbf{f}, \mathbf{u})] = \max_{\phi(\mathbf{u})} \mathcal{F}[\phi(\mathbf{u}) p(\mathbf{f} \mid \mathbf{u})] \tag{2.15}$$

$$\geq \mathcal{F}[\phi(\mathbf{u}) p(\mathbf{f} \mid \mathbf{u})] \tag{2.16}$$

This maximisation can be performed analytically and is detailed in Titsias (2009b, Appendix A). The key steps are:

$$\begin{aligned}\mathcal{F} &= \int q(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u})}{q(\mathbf{f}, \mathbf{u})} \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u} \\ &= \int q(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u})}{q(\mathbf{f}, \mathbf{u})} \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u} \\ &\overset{(2.12)}{=} p(\mathbf{f} \mid \mathbf{u}) \phi(\mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) \cancel{p(\mathbf{f} \mid \mathbf{u})} p(\mathbf{u})}{\cancel{p(\mathbf{f} \mid \mathbf{u})} \phi(\mathbf{u})} \mathrm{d}\mathbf{f}\, \mathrm{d}\mathbf{u} \\ &= \int \phi(\mathbf{u}) \left[\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) \mathrm{d}\mathbf{f} \right] \mathrm{d}\mathbf{u} + \underbrace{\int \phi(\mathbf{u}) \log \frac{p(\mathbf{u})}{\phi(\mathbf{u})} \mathrm{d}\mathbf{u}}_{= -\mathrm{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u}))}.\end{aligned} \tag{2.17}$$

The "problematic" (expensive to compute) conditionals $p(\mathbf{f} \mid \mathbf{u})$, which have been approximated in the previous methods, cancel.

The remaining integral w.r.t. $\mathbf{f}$ can be performed analytically, and the functional optimisation of the resulting terms with respect to $\phi(\mathbf{u})$ yields a Gaussian distribution

$$\phi_{\text{opt}}(\mathbf{u}) = \arg\max_{\phi(\mathbf{u})} \mathcal{F} = \mathcal{N}(\mu_\phi, \Sigma_\phi) \tag{2.18a}$$

$$\mu_\phi = \frac{1}{\sigma_n^2} K_{\mathbf{uu}} \Sigma \mathbf{y} \tag{2.18b}$$

$$\Sigma_\phi = K_{\mathbf{uu}} \Sigma K_{\mathbf{uu}} \tag{2.18c}$$

where $\Sigma = (K_{\mathbf{uu}} + \sigma_n^{-2} K_{\mathbf{uf}} K_{\mathbf{fu}})^{-1}$. Plugging these results into Equation (2.16) yields the optimal bound:

$$\mathcal{F}_{\text{opt}} = \log\left[\mathcal{N}\left(\mathbf{y} \mid 0, Q_{\mathbf{ff}} + \sigma_n^2 I\right)\right] - \frac{1}{2\sigma_n^2} \operatorname{tr}\left(K_{\mathbf{ff}} - Q_{\mathbf{ff}}\right). \tag{2.19}$$

Similar to FITC, we can use this bound to optimise both the hyperparameters $\boldsymbol{\theta}$ and the inducing input locations Z. In contrast to FITC, VFE is not prone to overfitting, as $\mathcal{F}_{\text{opt}}$ is a lower bound to the marginal likelihood of the full GP; VFE is therefore variationally protected from overfitting (see also Chapter 3).

The variational approximation of the prior $q(\mathbf{f}, \mathbf{u})$ is also used to compute the predictive distribution:

$$\begin{aligned}
p(\mathbf{f}^* \mid \mathbf{y}, X, X^*) &= \int p(\mathbf{f}^*, \mathbf{f}, \mathbf{u} \mid \mathbf{y}) \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{u} \\
&= \int p(\mathbf{f}^* \mid \mathbf{f}, \mathbf{u}) p(\mathbf{f}, \mathbf{u} \mid \mathbf{y}) \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{u} \\
&\overset{(a)}{\approx} \int p(\mathbf{f}^* \mid \mathbf{f}, \mathbf{u}) p(\mathbf{f} \mid \mathbf{u}) \phi_{\text{opt}}(\mathbf{u}) \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{u} \\
&\overset{(b)}{\approx} \int p(\mathbf{f}^* \mid \mathbf{u}) p(\mathbf{f} \mid \mathbf{u}) \phi_{\text{opt}}(\mathbf{u}) \, \mathrm{d}\mathbf{f} \, \mathrm{d}\mathbf{u} \\
&= \int p(\mathbf{f}^* \mid \mathbf{u}) \phi_{\text{opt}}(\mathbf{u}) \, \mathrm{d}\mathbf{u},
\end{aligned} \tag{2.20}$$

where, in addition to the variational approximation in step (a), we have also assumed conditional independence between training and test outputs given the inducing outputs in step (b), similar to Figure 2.2 *(right)*. The Gaussian integral yields the following predictive distribution:

$$p_{\text{VFE}}(\mathbf{f}^* \mid \mathbf{y}, X, X^*) = \mathcal{N}\left(\boldsymbol{\mu}_{\text{VFE}}, \Sigma_{\text{VFE}}\right) \tag{2.21a}$$

$$\boldsymbol{\mu}_{\text{VFE}} = \sigma^{-2} K_{*\mathbf{u}} (K_{\mathbf{uu}} + \sigma^{-2} K_{\mathbf{fu}} K_{\mathbf{fu}})^{-1} K_{\mathbf{uf}} \mathbf{y} \tag{2.21b}$$

$$\Sigma_{\text{VFE}} = K_{**} - Q_{**} + K_{*\mathbf{u}} (K_{\mathbf{uu}} + \sigma^{-2} K_{\mathbf{uf}} K_{\mathbf{fu}})^{-1} K_{\mathbf{u}*} \tag{2.21c}$$

This is exactly the same predictive distribution as for DTC (see Equation (2.6)). However, the objective function that is used to optimise the hyperparameters

$\boldsymbol{\theta}$ and the inducing input locations Z is different, compare Equation (2.7) and Equation (2.16). This is why VFE does not show the same overfitting behaviour as DTC. The additional trace term has already been introduced ad hoc by (Smola and Schölkopf, 2000) as a greedy selection criterion for a sparse approximation in kernel machines but has, to our knowledge, first been rigorously derived by (Titsias, 2009a). The trace term can be interpreted as the sum of the conditional variances of the training latent variables given the inducing variables. The importance of this term is discussed in Chapter 3.

Here we have presented the VFE approximation with the finite-dimensional Gaussian distributions instead of the infinite-dimensional Gaussian process, and have considered KL divergences between finite-dimensional distributions. Matthews et al. (2016) provide a mathematical discussion of these arguments in the infinite-dimensional case and how to treat the KL divergence between stochastic processes.

2.5 GAUSSIAN PROCESSES FOR BIG DATA AND GENERAL LIKELIHOODS

In the VFE approximation above we have collapsed the mean μ_ϕ and covariance Σ_ϕ of the variational distribution $\phi(\mathbf{u})$ analytically by optimising the evidence lower bound $\mathcal{F}$ (see Equation (2.18)). This has the advantage of reducing the number of variational parameters to the inducing input locations Z. However, it also has the disadvantage that it couples the observations and does not allow for minibatching of the objective function in Equation (2.19) (Hensman et al., 2013). Hensman et al. (2013) show that by *not* marginalising the inducing outputs $\mathbf{u}$, we can obtain an objective function that is accessible to stochastic optimisation through stochastic variational inference (SVI) (Hoffman et al., 2013). The cost of this is that we need to optimise the variational mean and covariance parameters of the inducing outputs as well.

Following Hensman et al. (2013), we use a Gaussian variational distribution with mean parameter $\mathbf{m}$ and covariance parameters $\mathbf{S}$

$$\phi(\mathbf{u}) = \mathcal{N}(\mathbf{u}; \mathbf{m}, \mathbf{S}) \,, \tag{2.22}$$

which gives rise to the looser ELBO

$$\begin{aligned} \mathcal{F} = \sum_{n=1}^{N} \left[\log \mathcal{N}(y_n; K_{\mathbf{f}_n\mathbf{u}} K_{\mathbf{uu}}^{-1}\mathbf{m}, \sigma^2 \mathbb{1}) - \tfrac{1}{2}\operatorname{tr}(\mathbf{S}\Lambda_n) - \tfrac{1}{2\sigma^2}\Sigma_{nn}\right] \\ - \operatorname{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u})) \end{aligned} \tag{2.23a}$$

$$\Lambda_n = \tfrac{1}{\sigma^2} K_{\mathbf{uu}}^{-1} K_{\mathbf{u}\mathbf{f}_n} K_{\mathbf{f}_n\mathbf{u}} K_{\mathbf{uu}}^{-1} \tag{2.23b}$$

$$\Sigma = K_{\mathbf{ff}} - K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} K_{\mathbf{uf}} \tag{2.23c}$$

and to the following approximate posterior predictive distribution:

$$p_{\text{SVI}}(\mathbf{f}^* \mid \mathbf{y}, X, X^*) = \mathcal{N}\left(\boldsymbol{\mu}_{\text{SVI}}, \Sigma_{\text{SVI}}\right) \tag{2.24a}$$

$$\boldsymbol{\mu}_{\text{SVI}} = K_{*\mathbf{u}} K_{\mathbf{uu}}^{-1} \mathbf{m} \tag{2.24b}$$

$$\Sigma_{\text{SVI}} = K_{**} - K_{*\mathbf{u}} K_{\mathbf{uu}}^{-1} (K_{\mathbf{uu}} - \mathbf{S}) K_{\mathbf{uu}}^{-1} K_{\mathbf{u}*} \,. \tag{2.24c}$$

We select the variational parameters by numerical maximisation of the ELBO using stochastic optimisation (Hensman et al., 2013). That is, to reduce the cost of evaluating the whole sum, we evaluate the bound stochastically by sub-sampling or minibatching. This also correctly minimises the KL divergence between the approximate and exact posteriors $\mathrm{KL}(q(f) \parallel p(f|\mathbf{y}))$ (Matthews et al., 2016) and allows Gaussian processes to be applied to large datasets.

The above derivation of the ELBO in Equation (2.23) relies on the conjugacy of the Gaussian likelihood. Hensman et al. (2015) discuss the case of more general likelihoods with a focus on classification.

For general likelihoods, we can express the corresponding ELBO by using the results in Equation (2.17)

$$\mathcal{F} = \int \phi(\mathbf{u}) \left[\int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) \, \mathrm{d}\mathbf{f} \right] \mathrm{d}\mathbf{u} - \mathrm{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u})) \tag{2.17}$$

$$= \mathbb{E}_{q(\mathbf{f})} \left[\log p(\mathbf{y} \mid \mathbf{f}) \right] - \mathrm{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u})) \,, \tag{2.25}$$

where $q(\mathbf{f})$ denotes the marginal

$$q(\mathbf{f}) = \int q(\mathbf{f}, \mathbf{u}) \, \mathrm{d}\mathbf{u} = \int p(\mathbf{f} \mid \mathbf{u}) \phi(\mathbf{u}) \, \mathrm{d}u \,. \tag{2.26}$$

For a Gaussian variational distribution (see Equation (2.22)) it is given by the posterior predictive in Equation (2.24) but evaluated at $\mathbf{f}$ instead of f_*

$$q(\mathbf{f}) = \mathcal{N}(\mathbf{f}; \boldsymbol{\mu}_{\text{SVI}}, \Sigma_{\text{SVI}}) \tag{2.27a}$$

$$\boldsymbol{\mu}_{\text{SVI}} = K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} \mathbf{m} \tag{2.27b}$$

$$\Sigma_{\text{SVI}} = K_{\mathbf{ff}} - K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} (K_{\mathbf{uu}} - \mathbf{S}) K_{\mathbf{uu}}^{-1} K_{\mathbf{uf}} \,. \tag{2.27c}$$

For likelihoods that decompose over the datapoints such as the Gaussian or classification likelihoods

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_{n=1}^{N} p(y_n \mid f_n) \tag{2.28}$$

the ELBO again decomposes into a sum over datapoints

$$\mathcal{F}_{\text{SVI}} = \sum_{n=1}^{N} \mathbb{E}_{q(f_n)} \left[\log p(y_n \mid f_n)\right] - \mathrm{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u}))\,, \tag{2.29}$$

which is amenable to stochastic minibatch training; the intractable expectation in Equation (2.29) can be estimated by MC sampling or numerical integration (Hensman et al., 2015).

2.6 OTHER APPROXIMATIONS

For completeness we briefly mention several other popular approaches to approximate Gaussian processes that we do not use in this thesis.

Sparse spectral approximations

Sparse spectral approximations approximate the kernel in the Fourier domain as opposed to the real domain. This class of approximations only applies to stationary, i.e. shift-invariant, covariance functions, as it makes use of *Bochner's theorem* (Stein, 1999).

Most practical stationary covariance functions, such as the squared exponential, can be expressed as a Fourier transform of a corresponding *spectral density* $s_f^2 p_S(\mathbf{s})$:

$$k(\boldsymbol{\tau} = \mathbf{x} - \mathbf{x}') = \int s_f^2 p_S(\mathbf{s})\, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{\tau} \cdot \mathbf{s}}\, \mathrm{d}\mathbf{s}\,. \tag{2.30}$$

Spectral approximations approximate the continuous density $p_S(\mathbf{s})$ in the integral (Equation (2.30)) by a sum of δ-functions, thus, selecting a finite set of frequencies in a Monte Carlo fashion.

Lázaro-Gredilla et al. (2010) introduce Sparse Spectral Gaussian Processes (SSGP), in which they optimise these frequencies using the marginal likelihood. They show that this model is equivalent to Bayesian basis function regression with trigonometric basis functions, and can lead to good performance compared to FITC. However, this approach is also prone to overfitting by being overconfident, especially for a large number of spectral points, compared to the number of data points (Lázaro-Gredilla et al., 2010). They propose to diagnose overfitting by widely varying predictive distributions for different initial conditions.

Random Fourier feature methods choose the frequencies randomly and have been referred to as *Random Kitchen Sinks* (RKS) in the kernel machines literature (Rahimi and Recht, 2008, 2009). For a more general discussion to approximate p_S in Equation (2.30) by a mixture of a continuous and a discrete density refer to Samo and Roberts (2015). Recently, several extensions to RKS have been proposed that also address GP regression and further speed up RKS using structured matrices for fast matrix vector multiplication. These include Fastfood (Le et al.,

2013), à la carte (Yang et al., 2015), or Extended and Unscented Kitchen Sinks (Bonilla et al., 2016).

Structured matrix approximation

Structured matrix approximations utilise structure in the covariance matrix that allows for fast computation of matrix inverses; while inversion and the eigendecomposition of a general $N \times N$ matrix scale cubically with N, faster algorithms exist if the matrix has structure. For example, when the covariance function factorises over dimensions or groups of dimensions, the kernel matrix has a kronecker structure over these groups and their eigendecomposition can be computed separately (Saatçi, 2011; Wilson et al., 2014). Similarly, when evaluating a stationary covariance function on a regular 1D grid, the resulting covariance matrix will have Toeplitz structure, which allows for $\mathcal{O}(N^2)$ inversion (Storkey, 1999; Yunong Zhang et al., 2005; Cunningham et al., 2008). Wilson and Nickisch (2015) unify both of these approaches in the KISS-GP approximation in which they use sparse approximations introduced above but place the inducing inputs on a regular grid to obtain structured matrices. Wilson et al. (2016b,a) use these approximations to present deep kernel learning, an approach to learn flexible kernels.

3

UNDERSTANDING PROBABILISTIC SPARSE GAUSSIAN PROCESS APPROXIMATIONS

In this chapter, we aim to thoroughly investigate and characterise the difference in behaviour of two popular sparse probabilistic Gaussian process approximations: FITC (Section 2.3) and VFE (Section 2.4). We investigate the biases of their objectives when learning hyperparameters, how and where each method allocates its modelling capacity, and their optimisation behaviour. We discuss the theoretical and practical properties of the two approaches. Our aim is to understand the approximations in detail, in order to know under which conditions each method is likely to succeed or fail in practice. We highlight issues which may arise in practical situations, and how to diagnose and mitigate them. Some of the properties of the methods have been previously reported in the literature, our aim here is a more complete and comparative approach.

This chapter is based on the conference paper 'Understanding Probabilistic Sparse Gaussian Process Approximations' (Bauer et al., 2016); it is joint work with Mark van der Wilk and Carl E. Rasmussen. My main contributions were to jointly develop the idea and independently devise and perform the mathematical analysis and experiments.

Throughout this chapter, we use the 1D toy-dataset by Snelson and Ghahramani (2006) as a running example to illustrate our findings. We focus on models with Gaussian likelihood and choose the squared exponential automatic relevance detection (ARD) covariance function (Neal, 1994; MacKay, 1992) that is widely used in practice:

$$k_{\text{ARD}}(\mathbf{x}, \mathbf{x}') = s_f \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^\intercal \Lambda^{-1}(\mathbf{x}-\mathbf{x}')\right) \tag{3.1}$$

where $\Lambda = \text{diag}(\ell_1^2, \ldots, \ell_d^2)$ is a diagonal $d \times d$-matrix of squared lengthscales. If a dimension i is non-informative, its associated lengthscale ℓ_i can be set to a large value, such that this dimension no longer contributes to the covariance. In 1D the ARD covariance function coincides with the standard squared exponential function.

3.1 OBJECTIVE FUNCTION FOR PROBABILISTIC SPARSE GP APPROXIMATIONS

Because of the similarity between the VFE and FITC objective (see Equations (2.10) and (2.19)) we introduce a common notation for their respective negative log marginal likelihood (NLML), which we minimise to train the methods

$$\mathcal{F} = \frac{N}{2}\log(2\pi) + \underbrace{\frac{1}{2}\log|Q_{\mathbf{ff}} + G|}_{\text{complexity penalty}} + \underbrace{\frac{1}{2}\mathbf{y}^\intercal(Q_{\mathbf{ff}} + G)^{-1}\mathbf{y}}_{\text{data fit}} + \underbrace{\frac{1}{2\sigma_n^2}\operatorname{tr}(T)}_{\text{trace term}}, \tag{3.2}$$

where

$$G_{\text{FITC}} = \operatorname{diag}[K_{\mathbf{ff}} - Q_{\mathbf{ff}}] + \sigma_n^2 I \qquad G_{\text{VFE}} = \sigma_n^2 I \tag{3.3}$$

$$T_{\text{FITC}} = 0 \qquad T_{\text{VFE}} = K_{\mathbf{ff}} - Q_{\mathbf{ff}}. \tag{3.4}$$

The common objective function has three terms, (i) a data fit term, (ii) a complexity penalty, and (iii) a trace term. Out of these, only the data fit and complexity penalty have direct analogues in the log marginal likelihood of a full GP model (refer to Equation (1.14)).

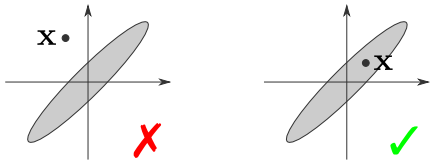
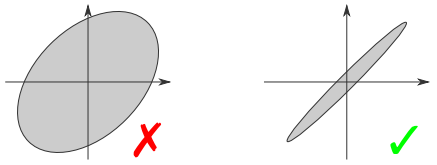
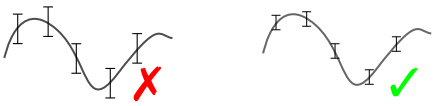
Term	Preference	Present in VFE	Present in FITC
Data fit $\frac{1}{2}\mathbf{y}^\intercal(Q_{\mathbf{ff}} + G)^{-1}\mathbf{y}$		✓	✓
Complexity $\frac{1}{2}\log\lvert Q_{\mathbf{ff}} + G\rvert$		✓	✓
Trace $\frac{1}{2\sigma_n^2}\operatorname{tr}(T)$		✓	✗

Figure 3.1: Sketch of configurations preferred by the individual terms of the objective function Equation (3.2)

data fit term

The *data fit* term penalises the data lying outside the covariance ellipse $Q_{\mathbf{ff}} + G$, see Figure 3.1 top row.

complexity penalty

The *complexity penalty* is the integral of the data fit term over all possible observations $\mathbf{y}$. It characterises the *volume* of possible datasets that are compatible with the data fit term. This can be seen as an instance of *Occam's razor* (see the discussion in Section 1.2), by penalising the methods for being able to predict too many datasets, see Figure 3.1 middle row.

VFE's trace term

The *trace term* in VFE ensures that the objective function is a lower bound to the marginal likelihood of the full GP. Without this term, VFE is identical to the earlier DTC approximation (Seeger et al., 2003) which can grossly over-estimate the marginal likelihood. The trace term penalises the sum of the conditional variances at the training inputs, conditioned on the inducing inputs (Titsias, 2009b), see Figure 3.1 bottom row. Intuitively, it ensures that VFE not only models this specific dataset $\mathbf{y}$ well, but also approximates the covariance structure of the full GP, $K_{\mathbf{ff}}$.

3.2 FITC CAN SEVERELY UNDERESTIMATE THE NOISE VARIANCE, VFE OVERESTIMATES IT

homoscedastic noise

heteroscedastic noise

In the full GP model with Gaussian likelihood we assume a homoscedastic (input independent) noise model with noise variance parameter σ_n^2. It fully characterises the uncertainty left after completely learning the latent function. In this section we show how FITC can use the diagonal term $\mathrm{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}})$ in G_{FITC} as additional heteroscedastic (input dependent) noise (Snelson and Ghahramani, 2006) to account for these differences, invalidating the above interpretation of the noise variance parameter. In fact, the FITC objective function encourages underestimation of the noise variance, whereas the VFE bound encourages overestimation. The latter is in line with previously reported biases of variational methods (Turner and Sahani, 2011).

Figure 3.2 shows the configuration most preferred by the FITC objective for a subset of 100 data points of the Snelson dataset. These were found by a thorough search for a minimum over hyperparameters, inducing inputs and number of inducing points. The noise is shrunk to practically zero, despite the mean prediction not going through every data point. Note how the learned mean still behaves well and how the training data lie well within the predictive variance. Only when considering predictive probabilities will this behaviour cause diminished performance. VFE, on the other hand, is able to approximate the posterior predictive distribution almost exactly.

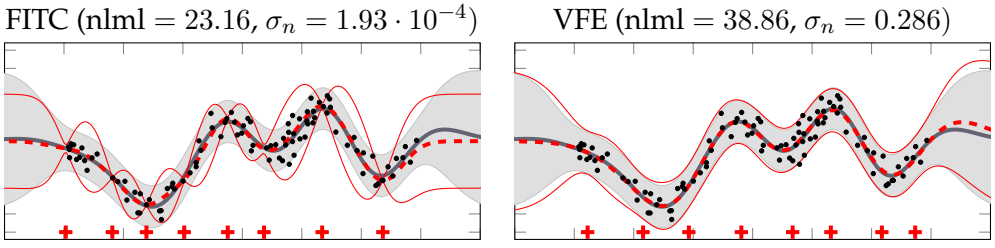


Figure 3.2: Behaviour of FITC and VFE on subset of 100 data points of the Snelson dataset for 8 inducing inputs (red crosses indicate inducing inputs; red lines indicate mean and 2σ) compared to the prediction of the full GP in grey. Optimised values for the full GP: nlml $= 34.15$, $\sigma_n = 0.274$

For both approximations, the complexity penalty decreases with decreased noise variance, by reducing the volume of datasets that can be explained. However, for a full GP and VFE this is accompanied by a data fit penalty for data points lying far away from the predictive mean. FITC, on the other hand, has an additional mechanism to avoid this penalty: its diagonal correction term $\mathrm{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}})$. This term can be seen as an input dependent or heteroscedastic noise term (discussed as a modelling advantage by Snelson and Ghahramani (2006)), which is zero exactly at an inducing input, and which grows to the prior variance away from an inducing input. By placing the inducing inputs near training data that happen to lie near the mean, the heteroscedastic noise term is locally shrunk, resulting in a reduced complexity penalty. Data points both far from the mean and far from inducing inputs do not incur a data fit penalty, as the heteroscedastic noise term has increased around these points. This mechanism removes the need for the homoscedastic noise to explain deviations from the mean, such that σ_n^2 can be turned down to reduce the complexity penalty further.

This explains the extreme pinching (severely reduced noise variance) observed in Figure 3.2. In examples with more densely packed data, there may not be any places where a near-zero noise point can be placed without incurring a huge data-fit penalty. However, inducing inputs will be placed where the data happens to randomly cluster around the mean, which still results in a decreased (albeit less extreme) noise estimate.

Remark 1 *FITC has alternative mechanisms to explain deviations from the learned function than the likelihood noise and will underestimate σ_n^2 as a consequence. In extreme cases, σ_n^2 can incorrectly be estimated to be almost zero.*

As a consequence of this additional mechanism, σ_n^2 can no longer be interpreted in the same way as for VFE or the full GP. σ_n^2 is often interpreted as the amount of uncertainty in the dataset which cannot be explained. Based on this interpretation, a low σ_n^2 is often used as an indication that the dataset is being fitted well. Active learning applications rely on a similar interpretation to differentiate between inherent noise, and uncertainty in the latent GP which can be reduced. FITC's different interpretation of σ_n^2 will cause efforts like these to fail.

VFE, on the other hand, is biased towards over-estimating the noise variance, because of both the data fit and the trace term. $Q_{\mathbf{ff}} + \sigma_n^2 I$ has $N - M$ eigenvectors with an eigenvalue of σ_n^2, since the rank of $Q_{\mathbf{ff}}$ is M. Any component of $\mathbf{y}$ in these directions will result in a larger data fit penalty than for $K_{\mathbf{ff}}$, which can only be reduced by increasing σ_n^2. The trace term can also be reduced by increasing σ_n^2.

Remark 2 *The VFE objective tends to over-estimate the noise variance compared to the full GP.*

3.3 VFE IMPROVES WITH ADDITIONAL INDUCING INPUTS, FITC MAY IGNORE THEM

Here we investigate the behaviour of each method when more inducing inputs are added. For each method, adding an extra inducing input gives it an extra basis function to model the data with. We discuss how and why VFE always improves, while FITC may deteriorate.

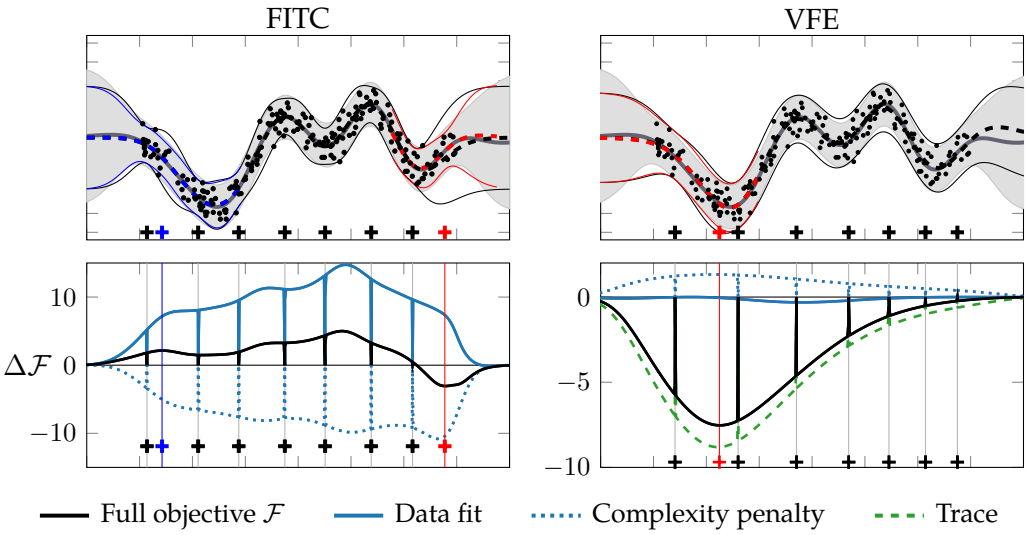


Figure 3.3: Change of the objective $\Delta\mathcal{F}$ and the predictive distribution when adding additional inducing inputs. *Top:* Predictive means and standard deviations for FITC and VFE for 7 optimised inducing inputs (black **+**) and how they change when a new inducing input is added at one of two possible locations (**+** or **+**). *Bottom:* Change in objective function from adding an inducing input anywhere along the x-axis. The overall change is decomposed into the change in the individual terms of Equation (3.2) (see legend). Two particular additional inducing inputs and their effect on the predictive distribution are shown in blue **+** and red **+**.

Figure 3.3 shows an example of how the objective function changes when an inducing input is added anywhere in the input domain. While the change in objective function $\Delta\mathcal{F}$ looks smooth overall, there are pronounced spikes for both, FITC and VFE. These return the objective to the value without the additional inducing input and occur at the locations of existing inducing inputs. We discuss the general change first before explaining the spikes.

We show in Lemma 3.1 that adding an inducing input corresponds to a rank-1 update of $Q_{\mathbf{ff}}$, and in Proposition 3.1 we prove that this always improves the VFE

bound[1]. VFE's complexity penalty increases due to an extra non-zero eigenvalue in $Q_{\mathbf{ff}}$, but gains in data fit and trace.

Remark 3 *VFE's posterior and marginal likelihood approximation become more accurate (or remain unchanged) regardless of where a new inducing input is placed.*

Lemma 3.1

Adding an inducing input corresponds to a rank-1 update for the approximate covariance matrix $Q_{\mathbf{ff}} = K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}K_{\mathbf{uf}}$ *of the form* $Q_{\mathbf{ff}}^{+} = Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^{\intercal}$ *with* $\mathbf{b} = \frac{1}{\sqrt{c}}(K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}k_{\mathbf{u}} - k_{\mathbf{f}})$.

Proof.

Let $K_{\mathbf{uu}}$ denote the covariance matrix of the M inducing inputs. We then add a new $M+1$st inducing input and denote all quantities depending on the new set of $M+1$ inducing inputs by a superscript $+$.

The updated approximate covariance matrix $Q_{\mathbf{ff}}^{+}$ is then given by:

$$Q_{\mathbf{ff}}^{+} = K_{\mathbf{fu}}^{+}(K_{\mathbf{uu}}^{+})^{-1}K_{\mathbf{uf}}^{+} \tag{3.5}$$

We proceed by first computing an explicit expression for the $M+1 \times M+1$ matrix $(K_{\mathbf{uu}}^{+})^{-1}$ before computing $Q_{\mathbf{ff}}^{+}$. For this we employ the block matrix inversion formula[2].

$$\begin{aligned}(K_{\mathbf{uu}}^{+})^{-1} &= \begin{pmatrix} K_{\mathbf{uu}} & k_{\mathbf{u}} \\ k_{\mathbf{u}}^{\intercal} & k \end{pmatrix}^{-1} \\ &= \begin{pmatrix} K_{\mathbf{uu}}^{-1} + \frac{1}{c}\mathbf{a}\mathbf{a}^{\intercal} & -\frac{1}{c}\mathbf{a} \\ -\frac{1}{c}\mathbf{a}^{\intercal} & \frac{1}{c} \end{pmatrix}\end{aligned}$$

where $\mathbf{a} = K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}$, $c = k - k_{\mathbf{u}}^{\intercal}K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}$ is the Schur complement of $K_{\mathbf{uu}}$, $k_{\mathbf{u}}^{\intercal} = (k(\mathbf{z}_1, \mathbf{z}_{M+1}), \ldots, k(\mathbf{z}_M, \mathbf{z}_{M+1}))$ is the vector of covariances between the old inducing inputs and the added inducing input, and $k = k(\mathbf{z}_{M+1}, \mathbf{z}_{M+1})$ is the covariance function evaluated at the new inducing input. Note that c needs to be non-zero for this expression to be well-defined.

$$K_{\mathbf{uf}}^{+} = \begin{pmatrix} K_{\mathbf{uf}} \\ k_{\mathbf{f}}^{\intercal} \end{pmatrix},$$

where $k_{\mathbf{f}}^{\intercal} = (k(\mathbf{z}_{M+1}, x_1), \ldots, k(\mathbf{z}_{M+1}, x_N))$, is the vector of covariances between the data points and the new inducing input.

1 Matthews (2016) independently proved this result by considering the KL divergence between processes. Titsias (2009a) proved this result for the special case when the new inducing input is selected from the training data.

2 https://en.wikipedia.org/wiki/Block_matrix#Block_matrix_inversion

We can now compute $Q_{\mathbf{ff}}^+$ as defined in Equation (3.5) and find that it is indeed given by a rank-1 update of $Q_{\mathbf{ff}}$

$$\begin{aligned}
Q_{\mathbf{ff}}^+ &= K_{\mathbf{fu}}^+ (K_{\mathbf{uu}}^+)^{-1} K_{\mathbf{uf}}^+ \\
&= K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} K_{\mathbf{uf}} + \frac{1}{c} \left(K_{\mathbf{fu}} \mathbf{a}\mathbf{a}^\intercal K_{\mathbf{uf}} + K_{\mathbf{fu}} \mathbf{a} \mathbf{a}^\intercal K_{\mathbf{uf}} \right. \\
&\qquad \left. - K_{\mathbf{fu}} \mathbf{a} k_{\mathbf{f}}^\intercal - k_{\mathbf{f}} \mathbf{a}^\intercal K_{\mathbf{uf}} + k_{\mathbf{f}} k_{\mathbf{f}}^\intercal \right) \\
&= K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} K_{\mathbf{uf}} + \frac{1}{c} \left(K_{\mathbf{fu}} \mathbf{a} - k_{\mathbf{f}} \right) \left(K_{\mathbf{fu}} \mathbf{a} - k_{\mathbf{f}} \right)^\intercal \\
&= K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} K_{\mathbf{uf}} + \mathbf{b}\mathbf{b}^\intercal \\
&= Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^\intercal \\
Q_{\mathbf{ff}}^+ &= Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^\intercal,
\end{aligned}$$

where we have introduced the rank-1 update vector $\mathbf{b} = \frac{1}{\sqrt{c}} (K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} k_{\mathbf{u}} - k_{\mathbf{f}})$.

While this result has been derived for the case of no jitter, it also naturally extends to the case with finite jitter, which is then absorbed into the definition of $K_{\mathbf{uu}}$ and k, respectively. □

Proposition 3.1

The VFE objective Equation (3.2) always improves when adding additional inducing inputs (or stays the same in the worst case).

Proof.

Using Lemma 3.1 we can compute the change in objective function when adding the $M+1$ st inducing input:

$$\begin{aligned}
2(\mathcal{F}^+ - \mathcal{F}) &= \log \left| Q_{\mathbf{ff}}^+ + \sigma_n^2 I \right| - \log \left| Q_{\mathbf{ff}} + \sigma_n^2 I \right| \\
&\quad + \mathbf{y}^\intercal (Q_{\mathbf{ff}}^+ + \sigma_n^2 I)^{-1} \mathbf{y} - \mathbf{y}^\intercal (Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1} \mathbf{y} \\
&\quad + \frac{1}{\sigma_n^2} \operatorname{tr}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}^+) - \frac{1}{\sigma_n^2} \operatorname{tr}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}) \\
&= \log \left| Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^\intercal + \sigma_n^2 I \right| - \log \left| Q_{\mathbf{ff}} + \sigma_n^2 I \right| \\
&\quad + \mathbf{y}^\intercal (Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^\intercal + \sigma_n^2 I)^{-1} \mathbf{y} - \mathbf{y}^\intercal (Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1} \mathbf{y} \\
&\quad - \frac{1}{\sigma_n^2} \operatorname{tr}(\mathbf{b}\mathbf{b}^\intercal)
\end{aligned}$$

To deal with the log-determinant terms and the inverses, we employ the Matrix determinant lemma[3] and the Sherman–Morrison formula[4], respectively

$$\begin{aligned}
2(\mathcal{F}^+ - \mathcal{F}) &= \log(1 + \mathbf{b}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}) + \log\left|Q_{\mathbf{ff}} + \sigma_n^2 I\right| - \log\left|Q_{\mathbf{ff}} + \sigma_n^2 I\right| \\
&\quad + \mathbf{y}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{y} - \mathbf{y}^\intercal \frac{(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}\mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}}{1 + \mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}}\mathbf{y} \\
&\quad - \mathbf{y}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{y} + \frac{1}{\sigma_n^2}\operatorname{tr}(\mathbf{b}\mathbf{b}^\intercal) \\
&= \log(1 + \mathbf{b}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}) - \frac{1}{\sigma_n^2}\operatorname{tr}(\mathbf{b}\mathbf{b}^\intercal) \\
&\quad - \mathbf{y}^\intercal \frac{(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}\mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}}{1 + \mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}}\mathbf{y}
\end{aligned}$$

We can bound the first two terms by noting

$$\begin{aligned}
\operatorname{tr}(\mathbf{b}\mathbf{b}^\intercal) &= \mathbf{b}^\intercal\mathbf{b} \\
\log(1 + x) &\leq x \\
\mathbf{b}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b} &\leq \frac{1}{\sigma_n^2}\mathbf{b}^\intercal\mathbf{b}
\end{aligned}$$

Thus,

$$\log(1 + \mathbf{b}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}) - \frac{1}{\sigma_n^2}\operatorname{tr}(\mathbf{b}\mathbf{b}^\intercal) \leq 0$$

and equality holds for $\mathbf{b} = 0$, as is the case when both inducing inputs lie on top of each other. It remains to show that the term including the $\mathbf{y}$ (including its sign) is non-positive. This can be shown quite easily:

$$\begin{aligned}
-\mathbf{y}^\intercal \frac{(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}\mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}}{1 + \mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}}\mathbf{y} &= -\frac{(\mathbf{y}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b})^2}{1 + \mathbf{b}^T(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b}} \\
&\leq -(\mathbf{y}^\intercal(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}\mathbf{b})^2 \\
&\leq 0
\end{aligned}$$

where the second to last inequality holds as $(Q_{\mathbf{ff}} + \sigma_n^2 I)^{-1}$ is positive definite. Equalities hold, again, if $\mathbf{b} = 0$ which corresponds to duplication of an existing inducing input. □

We note that Proposition 3.1 does not apply to the more recent variational bound by (Hensman et al., 2013), in which the distribution over the inducing outputs is also optimised.

3 https://en.wikipedia.org/wiki/Matrix_determinant_lemma
4 https://en.wikipedia.org/wiki/Sherman-Morrison_formula

While the FITC objective function can change either way, the heteroscedastic noise, which is given by $\operatorname{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}})$ always decreases or remains the same when a new inducing input is added:

$$\operatorname{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}^{+}) = \operatorname{diag}(K_{\mathbf{ff}} - (Q_{\mathbf{ff}} + \mathbf{b}\mathbf{b}^{\mathsf{T}})) \tag{3.6}$$

$$= \operatorname{diag}(K_{\mathbf{ff}} - Q_{\mathbf{ff}}) - \operatorname{diag}(\mathbf{b}\mathbf{b}^{\mathsf{T}}) \tag{3.7}$$

The diagonal elements of $\mathbf{b}\mathbf{b}^{\mathsf{T}}$ are given by b_m^2, which are always larger or equal to zero, such that the heteroscedastic noise always decreases (or stays the same).

For a squared exponential kernel, the decrease is strongest around the newly placed inducing input. This decrease has two effects. Firstly, it reduces the complexity penalty since the diagonal component of $Q_{\mathbf{ff}} + G$ is reduced and replaced by a more strongly correlated $Q_{\mathbf{ff}}$. Secondly, it worsens the data fit term as the heteroscedastic term is required to fit the data when the homoscedastic noise is underestimated. Figure 3.3 shows reduced error bars with several data points now outside of the 95% prediction bars. Also shown is a case where an additional inducing input improves the objective, where the extra correlations outweigh the reduced heteroscedastic noise.

Both VFE and FITC exhibit pathological behaviour (spikes) when inducing inputs are clumped, that is, when they are placed exactly on top of each other. In this case, the objective function has the same value as when all duplicate inducing inputs were removed as we show in Proposition 3.2. In other words, for all practical purposes, a model with duplicate inducing inputs reduces to a model with fewer, individually placed inducing inputs.

Proposition 3.2

A duplicate inducing input does not change the approximate covariance matrix, $Q_{\mathbf{ff}}^{+} = Q_{\mathbf{ff}}$

Proof.

One might be tempted to use Lemma 3.1 to evaluate the update when placing an additional inducing input on top of an existing one. In that case $K_{\mathbf{uu}}^{-1} k_{\mathbf{u}} = \widehat{e}_M$, where $\widehat{e}_M$ denotes the indicator vector with a one at the Mth position and zeros otherwise, and $K_{\mathbf{fu}} K_{\mathbf{uu}}^{-1} k_{\mathbf{u}} = k_{\mathbf{f}}$ suggesting $\mathbf{b} = 0$. However, in this case, the Schur complement that appears in the rank-1 update vector $\mathbf{b} \propto 1/\sqrt{c}$ vanishes as well, $c = 0$.

In the following we show that this reasoning can be made exact by considering a finite *jitter* term ϵI added onto $K_{\mathbf{uu}}^{+}$ before inversion. The result can be expanded to second order in ϵ and the limit $\epsilon \to 0$ then leads to the desired result. The intuition behind the fact that Equation (3.5) is well behaved, even if $K_{\mathbf{uu}}^{+}$ is singular, is, that the eigenvector of $K_{\mathbf{uu}}^{+}$ that corresponds to the zero eigenvalue is never excited by the matrix $K_{\mathbf{uf}}^{+}$ which has a duplicate row. The eigenvector only has two non-zero elements, which have the same absolute

value but different signs, thus cancelling with the duplicate rows in $K_{\mathbf{uf}}^{+}$. Moreover, we obtain a correction term that scales with the jitter. For reasons of numerical stability, one has to employ some form of regularisation of the (possibly) singular matrix $K_{\mathbf{uu}}$ in practise. One common method that is implemented in many toolboxes is the constant jitter ϵI introduced above. We assume that the original $K_{\mathbf{uu}}$ is non-singular for now.

We again employ the block matrix inversion formula[5] of the following form:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BF^{-1}CA^{-1} & -A^{-1}BF^{-1} \\ -F^{-1}CA^{-1} & F^{-1} \end{pmatrix} \tag{3.8}$$

where $F = D - CA^{-1}B$ is the Schur complement of A.

$$(K_{\mathbf{uu}}^{+} + \epsilon I_{M+1\times M+1})^{-1} = \begin{pmatrix} K_{\mathbf{uu}} + \epsilon I_{M\times M} & k_{\mathbf{u}} \\ k_{\mathbf{u}}^{\intercal} & k + \epsilon \end{pmatrix}^{-1} \tag{3.9}$$

where $k_{\mathbf{u}}^{\intercal} = (k(\mathbf{z}_1, \mathbf{z}_M), k(\mathbf{z}_2, \mathbf{z}_M), \ldots, k(\mathbf{z}_M, \mathbf{z}_M))$ and $k = k(\mathbf{z}_M, \mathbf{z}_M)$ similarly to before. In order to perform the inversion, we expand the inverses in Equation (3.8) to second order in ϵ:

$$\begin{aligned}
(K_{\mathbf{uu}} + \epsilon I)^{-1} &\approx K_{\mathbf{uu}}^{-1}(1 - \epsilon K_{\mathbf{uu}}^{-1} + \epsilon^2 K_{\mathbf{uu}}^{-2} - \epsilon^3 K_{\mathbf{uu}}^{-3}) && A^{-1} \\
(K_{\mathbf{uu}} + \epsilon I)^{-1} k_{\mathbf{u}} &\approx \widehat{e}_M - \epsilon k_{\mathbf{u}}^{-1} + \epsilon^2 K_{\mathbf{uu}}^{-1} k_{\mathbf{u}}^{-1} - \epsilon^3 K_{\mathbf{uu}}^{-2} k_{\mathbf{u}}^{-1} && A^{-1}B \\
k_{\mathbf{u}}^{T}(K_{\mathbf{uu}} + \epsilon I)^{-1} &\approx \widehat{e}_M^{T} - \epsilon (k_{\mathbf{u}}^{-1})^{T} + \epsilon^2 (k_{\mathbf{u}}^{-1})^{T} K_{\mathbf{uu}}^{-1} - \epsilon^3 (k_{\mathbf{u}}^{-1})^{T} K_{\mathbf{uu}}^{-2} && CA^{-1} \\
k_{\mathbf{u}}^{T}(K_{\mathbf{uu}} + \epsilon)^{-1} k_{\mathbf{u}} &\approx k - \epsilon + \epsilon^2 k^{-1} - \epsilon^3 (k_{\mathbf{u}}^{-1})^{T} k_{\mathbf{u}}^{-1} && CA^{-1}B
\end{aligned}$$

and compute the Schur complement and its inverse also to $\mathcal{O}(\epsilon^2)$

$$\begin{aligned}
(k + \epsilon) - k_{\mathbf{u}}^{T}(K_{\mathbf{uu}} + \epsilon)^{-1} k_{\mathbf{u}} &\approx 2\epsilon(1 - \frac{\epsilon}{2} k^{-1} + \frac{\epsilon^2}{2}(k_{\mathbf{u}}^{-1})^{T} k_{\mathbf{u}}^{-1}) \\
((k + \epsilon) - k_{\mathbf{u}}^{T}(K_{\mathbf{uu}} + \epsilon)^{-1} k_{\mathbf{u}})^{-1} &\approx \frac{1}{2\epsilon} + \frac{1}{4} k^{-1} - \frac{\epsilon}{4}(k_{\mathbf{u}}^{-1})^{T} k_{\mathbf{u}}^{-1} + \frac{\epsilon}{8}(k^{-1})^2
\end{aligned}$$

where $\widehat{e}_M = (0, \ldots, 0, 1)^{\intercal}$ is the indicator vector with a one at position M and $k_{\mathbf{u}}^{-1} = (k^{-1}(\mathbf{z}_1, \mathbf{z}_M), \ldots, k^{-1}(\mathbf{z}_M, \mathbf{z}_M))^{\intercal}$ is the Mth column of $K_{\mathbf{uu}}^{-1}$. Note that the elements of $k_{\mathbf{u}}^{-1}$ are not element-wise inverses but elements of an inverse matrix. Analogously, k^{-1} denotes the (M, M)th element of the matrix $K_{\mathbf{uu}}^{-1}$.

Plugging the individual terms into Equation (3.8), we now compute the inverse in Equation (3.9):

5 https://en.wikipedia.org/wiki/Block_matrix#Block_matrix_inversion

$$
\begin{aligned}
(K^{+}_{\mathbf{uu}}+\epsilon)^{-1} = &\left(\begin{array}{ccc|c} & K_{\mathbf{uu}}^{-1} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hline 0 & \cdots & 0 & 0 \end{array}\right) - \epsilon \left(\begin{array}{ccc|c} & K_{\mathbf{uu}}^{-2} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hline 0 & \cdots & 0 & 0 \end{array}\right) \\
&+ \frac{\epsilon}{2}\left(\begin{array}{ccc|c} & k_{\mathbf{u}}^{-1}(k_{\mathbf{u}}^{-1})^{\intercal} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hline 0 & \cdots & 0 & 0 \end{array}\right) + \frac{1}{2}\left(\begin{array}{ccc|cc} & & & 0 & 0 \\ & 0_{M-1\times M-1} & & \vdots & \vdots \\ & & & 0 & 0 \\ \hline 0 & \cdots & 0 & \epsilon^{-1} & -\epsilon^{-1} \\ \hline 0 & \cdots & 0 & -\epsilon^{-1} & \epsilon^{-1} \end{array}\right) \\
&+ \frac{1}{4}\left(\begin{array}{ccc|cc} & & & 0 & 0 \\ & 0_{M-1\times M-1} & & \vdots & \vdots \\ & & & 0 & 0 \\ \hline 0 & \cdots & 0 & k^{-1} & -k^{-1} \\ \hline 0 & \cdots & 0 & -k^{-1} & k^{-1} \end{array}\right) + \frac{1}{2}\left(\begin{array}{cc|c} 0_{M-1\times M-1} & -k_{u\setminus m}^{-1} & k_{\mathbf{u}}^{-1} \\ -(k_{u\setminus m}^{-1})^{T} & -2k^{-1} & \\ \hline (k_{\mathbf{u}}^{-1})^{T} & & 0 \end{array}\right) \\
&+ \frac{\epsilon}{4}\left(\begin{array}{cc|c} 0_{M-1\times M-1} & 2K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}^{-1} - k^{-1}k_{\mathbf{u}}^{-1} & -(2K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}^{-1} - k^{-1}k_{\mathbf{u}}^{-1}) \\ (2K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}^{-1} - k^{-1}k_{\mathbf{u}}^{-1})^{T} & 0 & \\ \hline -(2K_{\mathbf{uu}}^{-1}k_{\mathbf{u}}^{-1} - k^{-1}k_{\mathbf{u}}^{-1})^{T} & & 0 \end{array}\right) \\
&+ \frac{\epsilon}{8}\left(\begin{array}{ccc|c} & & & 0 \\ & 0_{M-1\times M-1} & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & (k^{-1})^2 - 2(k_{\mathbf{u}}^{-1})^{T}k_{\mathbf{u}}^{-1} \\ \hline 0 & \cdots & 0 & -((k^{-1})^2 - 2(k_{\mathbf{u}}^{-1})^{T}k_{\mathbf{u}}^{-1}) \end{array}\right. \\
&\left.\begin{array}{c|c} 0 & \\ \vdots & \\ 0 & \\ \hline -((k^{-1})^2 - 2(k_{\mathbf{u}}^{-1})^{T}k_{\mathbf{u}}^{-1}) & \\ \hline (k^{-1})^2 - 2(k_{\mathbf{u}}^{-1})^{T}k_{\mathbf{u}}^{-1} & \end{array}\right) + \mathcal{O}(\epsilon^2)
\end{aligned}
$$

When we now multiply out the product $K^{+}_{\mathbf{fu}}(K^{+}_{\mathbf{uu}}+\epsilon I)^{-1}K^{+}_{\mathbf{uf}}$, we note that $K^{+}_{\mathbf{uf}}$ will have a duplicate row and $K^{+}_{\mathbf{fu}}$ will have a duplicate column. Due to this, all terms that have the submatrix $0_{M-1\times M-1}$ in their upper left hand corner cancel. This includes the term that contains the diverging (in the limit $\epsilon \to 0$) inverse jitter ϵ^{-1}, and we are left with:

$$
\begin{aligned}
Q^{+}_{\mathbf{ff}} &= K^{+}_{\mathbf{fu}}(K^{+}_{\mathbf{uu}}+\epsilon I)^{-1}K^{+}_{\mathbf{uf}} \\
&= K_{\mathbf{fu}}K_{\mathbf{uu}}^{-1}K_{\mathbf{uf}} - \epsilon K_{\mathbf{fu}}K_{\mathbf{uu}}^{-2}K_{\mathbf{uf}} + \frac{\epsilon}{2}K_{\mathbf{fu}}k_{\mathbf{u}}^{-1}(k_{\mathbf{u}}^{-1})^{\intercal}K_{\mathbf{uf}} + \mathcal{O}(\epsilon^2) \\
&= Q_{\mathbf{ff}} - \epsilon K_{\mathbf{fu}}K_{\mathbf{uu}}^{-2}K_{\mathbf{uf}} + \frac{\epsilon}{2}K_{\mathbf{fu}}k_{\mathbf{u}}^{-1}(k_{\mathbf{u}}^{-1})^{\intercal}K_{\mathbf{uf}} + \mathcal{O}(\epsilon^2)
\end{aligned}
$$

Such that the correction to the original approximate covariance matrix is given by:

$$
Q^{+}_{\mathbf{ff}} - Q_{\mathbf{ff}} = \frac{\epsilon}{2}K_{\mathbf{fu}}k_{\mathbf{u}}^{-1}(k_{\mathbf{u}}^{-1})^{\intercal}K_{\mathbf{uf}} + \mathcal{O}(\epsilon^2) \tag{3.10}
$$

We can now take the limit $\epsilon \to 0$ as all the "infinities" have cancelled above. For finite jitter, the correction term is again given by a rank-1 update to first order in ϵ. □

Theoretically, these spikes that return the objective function to its unaltered value only occur at individual points, such that no gradients towards or away from them could exist and they would never be encountered. In practice, however, these peaks are widened by a finite *jitter* that is added to $K_{\mathbf{uu}}$ to ensure it remains well conditioned enough to be invertible. This finite width provides the gradients that allow an optimiser to detect these configurations.

As VFE always improves with additional inducing inputs, these configurations must correspond to maxima of the optimisation surface and clumping of inducing inputs does not occur for VFE. For FITC, configurations with clumped inducing inputs can and often do correspond to minima of the optimisation surface. By placing them on top of each other, FITC can avoid the penalty of adding an extra inducing input and can gain the bonus from the heteroscedastic noise. Clumping therefore constitutes a mechanism that allows FITC to effectively remove inducing inputs at no cost. In practice we find that convergence towards these minima can be slow.

We illustrate this behaviour in Figure 3.4 for 15 randomly initialised inducing inputs. FITC places some of them exactly on top of each other, whereas VFE spreads them out and recovers the full GP well.

Remark 4 *In FITC, having a good approximation $Q_{\mathbf{ff}}$ to $K_{\mathbf{ff}}$ must be traded off with the gains coming from the heteroscedastic noise. FITC does not always favour a more accurate approximation to the GP.*

Remark 5 *FITC avoids losing the gains of the heteroscedastic noise by placing inducing inputs on top of each other, effectively removing them.*

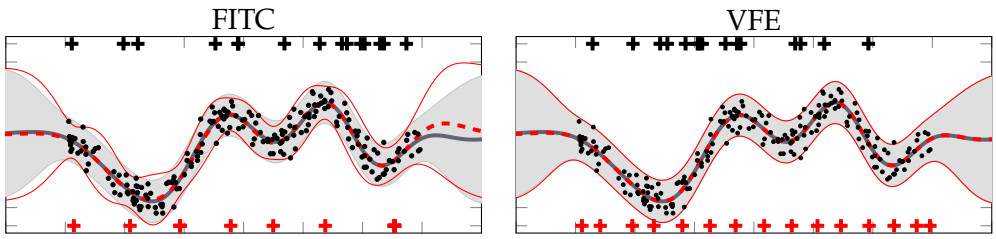


Figure 3.4: Predictive distributions for FITC and VFE with 15 inducing inputs (initial positions in black **+**, optimised positions in red **+**). Even following joint optimisation of inducing inputs and hyperparameters, FITC avoids the penalty of added inducing inputs by clumping some of them on top of each other (shown as a single red cross **+**). VFE spreads out the inducing inputs to get closer to the true full GP posterior.

Figure 3.5 shows the optimised location of the inducing inputs for a toy dataset with two-dimensional input (x_1, x_2). Datapoints are drawn from a GP with squared exponential kernel with isotropic lengthscale $\ell = 0.5$. The locations of the data inputs were chosen uniformly at random from $[-1, 1]^2$ but results were similar for inputs drawn from a Gaussian. We jointly optimise the hyperparameters and

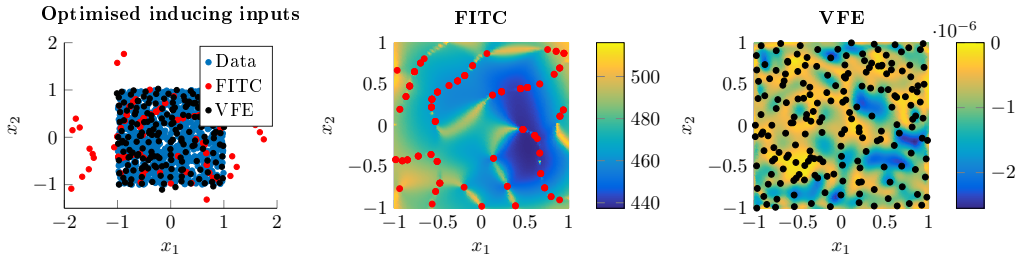


Figure 3.5: Clumping of inducing inputs for two dimensional input (x_1, x_2). *Left:* location of data points and optimised inducing inputs for FITC and VFE. FITC also tends to place inducing inputs in regions where no data points are located. *Middle and right:* Detail of *left* with optimised inducing inputs for FITC and VFE. The change of the objective function for placing an additional inducing input is shown as colourcode. FITC clumps the inducing inputs in filaments whereas VFE spreads them evenly. Moreover, VFE always improves when adding a new inducing input (negative values of change), whereas FITC always gets worse in this example (positive values of change)

the inducing input locations; the latter have been initialised to a subset of data. We make three observations that are in line with our previous observations and arguments: (i) VFE always gets better by placing an additional inducing input, whereas FITC can (and in this example does) get worse, see colour code in Figure 3.5 *middle* and *right*; (ii) FITC again clumps inducing inputs by arranging them in filaments, whereas VFE distributes them evenly; (iii) FITC places inducing inputs outside of the data input domain $[-1, 1]^2$. We have not discussed the placement of inducing inputs away from data inputs in detail; it can again be explained by FITC not being punished for heteroscedastic noise.

3.4 FITC DOES NOT RECOVER THE TRUE POSTERIOR, VFE DOES

In the previous section we showed that FITC has difficulty using additional resources to model the data, and that the optimiser moves away from a more accurate approximation to the full GP. This behaviour was first empirically reported by Matthews (2016, Sec. 4.6.1 and Fig. 4.2). Here we show that this behaviour is inherent to the FITC objective function.

Both VFE and FITC can recover the true posterior by placing an inducing input on every training input (Titsias, 2009a; Snelson, 2007). For VFE, this must be a global minimum, since the KL gap to the true marginal likelihood is zero. For FITC, however, this solution is merely a saddle point. The derivative of the inducing inputs is zero for this configuration, but the objective function can still be improved. As with the clumping behaviour, adding jitter subtly makes this behaviour more obvious by perturbing the gradients. In Figure 3.6 we reproduce the observations by Matthews (2016, Sec. 4.6.1 and Fig. 4.2) on the Snelson dataset for $N = 200$ data points and $M = 200$ inducing inputs that have been located on

the data inputs. VFE is at a minimum and does not move the inducing inputs, whereas FITC improves its objective and moves the inducing inputs considerably; see Figure 3.6 (left) for the improvement in objective function and Figure 3.6 (right) for the optimised inducing input locations.

	NLML	
Method	initial	final
Full GP	–	33.89
VFE	33.89	33.89
FITC	33.89	28.39

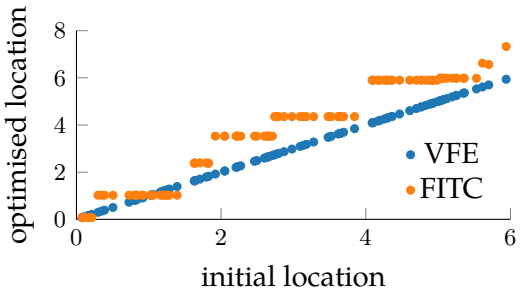


Figure 3.6: Results of optimising VFE and FITC after initialising the inducing inputs at the data points and the other hyperparameters at the solution that gives the correct posterior and marginal likelihood. We observe that FITC moves to a significantly different solution by choosing to cluster the inducing inputs, see optimised location *(right)*.

Remark 6 *FITC generally does not recover the full GP, even when it has enough resources.*

3.5 FITC RELIES ON LOCAL OPTIMA

So far, we have observed some cases where FITC fails to produce results in line with the full GP, and characterised why. In practice, FITC has performed well, and pathological behaviour is not always observed. In this section we discuss the optimiser dynamics and show that they help FITC behave reasonably.

To demonstrate this behaviour, we consider a 4d toy dataset: 1024 training and 1024 test samples drawn from a 4d Gaussian Process with isotropic squared exponential covariance function ($l = 1.5, s_f = 1$) and true noise variance $\sigma_n^2 = 0.01$. We fit both FITC and VFE to this dataset with the number of inducing inputs ranging from 16 to 1024, and compare them to the full GP in Figure 3.7.

VFE monotonically approaches the values of the full GP but initially overestimates the noise variance, as discussed in Section 3.2. Conversely, we can identify three regimes for the objective function of FITC: 1) Monotonic improvement for few inducing inputs, 2) a region where FITC over-estimates the marginal likelihood, and 3) recovery towards the full GP for many inducing inputs. Predictive performance follows a similar trend, first improving, then declining while the bound is estimated to be too high, followed by a recovery. The recovery is counter to the usual intuition that over-fitting worsens when adding more parameters.

We explain the behaviour in these three regimes as follows: When the number of inducing inputs are severely limited (regime 1), FITC needs to place them such

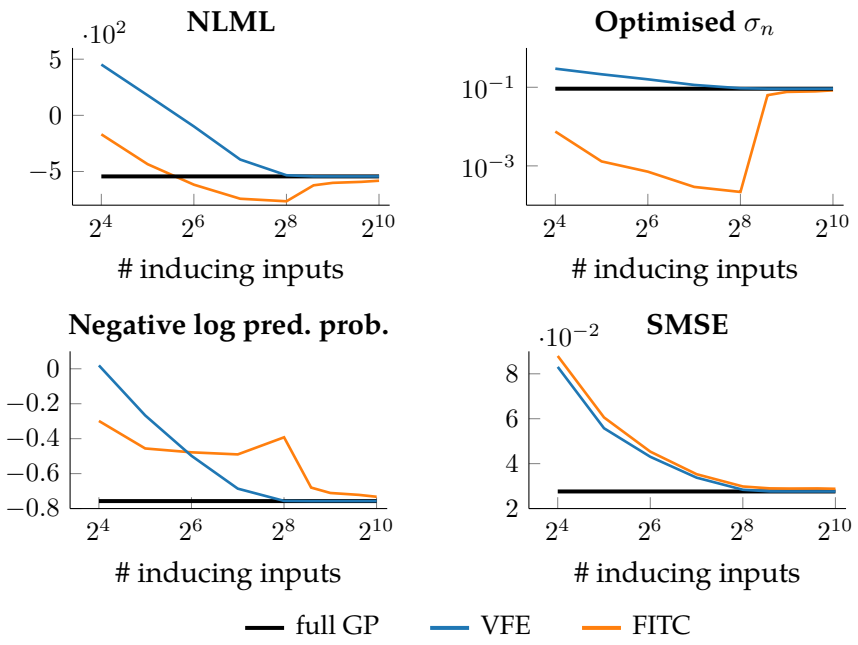


Figure 3.7: Optimisation behaviour of VFE and FITC for varying number of inducing inputs compared to the full GP. We show the objective function (negative log marginal likelihood), the optimised noise σ_n, the negative log predictive probability and standardised mean squared error as defined in Rasmussen and Williams (2006).

that $K_{\mathbf{ff}}$ is well approximated. This correlates most points to some degree, and ensures a reasonable data fit term. The marginal likelihood is under-estimated due to lack of a flexibility in $Q_{\mathbf{ff}}$.

As the number of inducing inputs increases (regime 2), the marginal likelihood is over-estimated and the noise drastically under-estimated. Additionally, performance in terms of log predictive probability deteriorates. This is the regime closest to FITC's behaviour in Figure 3.2. There are enough inducing inputs such that they can be placed so a bonus can be gained from the heteroscedastic noise, without gaining a complexity penalty from losing long scale correlations.

Finally, in regime 3, FITC starts to behave more like a regular GP in terms of marginal likelihood, predictive performance and noise variance parameter σ_n. FITC's ability to use heteroscedastic noise is reduced as the approximate covariance matrix $Q_{\mathbf{ff}}$ is closer to the true covariance matrix $K_{\mathbf{ff}}$ when many (initial) inducing inputs are spread over the input space.

In the previous section we showed that after adding a new inducing input, a better minimum, obtained without the extra inducing input, could be recovered by clumping. So it is clear that the minimum found with fewer active inducing inputs still exists in the optimisation surface of many inducing inputs; the optimiser just does not find it.

Remark 7 *When running FITC with many inducing inputs its resemblance to the full GP solution relies on local optima, rather than the objective function changing.*

3.6 VFE IS HINDERED BY LOCAL OPTIMA

So far we have seen that the VFE objective function is a true lower bound on the marginal likelihood and does not share the same pathologies as FITC. Thus, when optimising, we really are interested in finding a global optimum. The VFE objective function is not completely trivial to optimise, and tricks are often required to find a good optimum, such as initialising the inducing inputs with k-means and initially fixing the hyperparameters (Hensman et al., 2015, 2013). Others have commented that VFE has the tendency to underfit (Lázaro-Gredilla and Figueiras-Vidal, 2009). Here we investigate the underfitting claim and relate it to optimisation behaviour.

As this behaviour is not observable in our 1D dataset, we illustrate it on the pumadyn32nm dataset[6] (32 dimension, 7168 training, 1024 test), see Table 3.1 for the results of a representative run with random initial conditions and 40 inducing inputs.

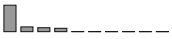
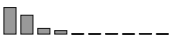
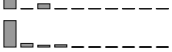
Method	**NLML**$/N$	σ_n	**inv. lengthscales**	**RMSE**
GP (SoD)	-0.099	0.196		0.209
FITC	-0.145	0.004		0.212
VFE	1.419	1		0.979
VFE (frozen)	0.151	0.278		0.276
VFE (init FITC)	-0.096	0.213		0.212

Table 3.1: Results for pumadyn32nm dataset. We show negative log marginal likelihood (NLML) divided by number of training points, the optimised noise variance σ_n^2, the ten most dominant inverse lengthscales and the RMSE on test data. Methods are full GP on 2048 training samples, FITC, VFE, VFE with initially frozen hyperparameters, VFE initialised with the solution obtained by FITC.

Using a squared exponential ARD kernel with separate lengthscales for every dimension, a full GP on a subset of data identified four lengthscales as important to model the data while scaling the other 28 lengthscales to large values (in the table we plot the inverse lengthscales).

FITC was consistently able to identify the same four lengthscales and performed similarly compared to the full GP but scaled down the noise variance σ_n^2 to almost zero. VFE, on the other hand, was unable to identify these relevant lengthscales when jointly optimising the hyperparameters and inducing inputs, and only identified some of the them when initially freezing the hyperparameters. One might say that VFE "underfits" in this case. However, we can show that VFE still

6 obtained from http://www.cs.toronto.edu/~delve/data/datasets.html

recognises a good solution: When we initialised VFE with the FITC solution it consistently obtained a good fit to the model with correctly identified lengthscales and a noise variance that was close to the full GP.

Remark 8 *VFE has a tendency to find under-fitting solutions. However, this is an optimisation issue. The bound correctly identifies good solutions.*

3.7 SUMMARY

In this chapter, we have thoroughly investigated and characterised the differences between FITC and VFE, both in terms of their objective function and their behaviour observed during practical optimisation. We highlighted several instances of undesirable behaviour in the FITC objective: over-estimation of the marginal likelihood, sometimes severe under-estimation of the noise variance parameter, wasting of modelling resources, and not recovering the true posterior. Moreover, the common practice of using the noise variance parameter as a diagnostic for good model fitting is unreliable.

In contrast, VFE is a true bound to the marginal likelihood of the full GP and behaves predictably: It correctly identifies good solutions, always improves with extra resources and recovers the true posterior when possible. In practice however, the pathologies of the FITC objective do not always show up, thanks to "good" local optima and (unintentional) early stopping. While VFE's objective recognises a good configuration, it is often more susceptible to local optima and harder to optimise than FITC.

However, based on the superior properties of the VFE objective function, we recommend using VFE, while paying attention to optimisation difficulties. These can be mitigated by careful initialisation, random restarts, other optimisation tricks and comparison to the FITC solution to guide VFE optimisation.

VFE's variational approach has been generalised to address a series of intractabilities and other computational issues such as minibatching of data (Hensman et al., 2013), scalable classification (Hensman et al., 2015), or convolutional GPs (van der Wilk et al., 2017), and we use it in the following chapter to incorporate general invariances into GP priors.

While the variational objective is generally well understood as a lower bound to the true marginal likelihood, several theoretical and practical aspects can still be explored. For example, the number and placement of inducing inputs as well as further discussion of optimisation difficulties and how to overcome them would be interesting research directions. Burt et al. (2019) very recently presented results in this vein.

4

LEARNING INVARIANCES USING THE MARGINAL LIKELIHOOD

Generalising well in supervised learning tasks relies on correctly extrapolating the training data to a large region of the input space. As we discussed in the introduction, many possible generalisations might be consistent with the data such that additional inductive biases are required to select certain generalisations over others. One way to achieve this is to constrain the predictions to be invariant to transformations of the input that are known to be irrelevant for the task at hand, for example, translations or small rotations. Commonly, this is done through data augmentation, where the training set is enlarged by applying hand-crafted transformations to the inputs.

In this chapter we argue that invariances should instead be incorporated in the model structure and priors, and learned using the marginal likelihood, which correctly rewards the reduced complexity of invariant models as discussed in Section 1.2. We demonstrate this for Gaussian process models, because of the ease with which their marginal likelihood can be estimated. Our main technical contribution is a variational inference scheme for Gaussian processes containing invariances described by a sampling procedure. Similar to data augmentation, this procedure samples transformed input images. However, instead of including them in the training set, they are used to construct an invariant covariance function, such that the marginal likelihood can be meaningfully estimated. We can then learn the properties of this sampling procedure by backpropagating through it to maximise the marginal likelihood, similar to the kernel hyperparameters. The inference scheme builds on top of the sparse variational approximation introduced in Chapter 2 and discussed in Chapter 3, as well as recent advances by van der Wilk et al. (2017).

The work presented in this chapter is based on joint research with Mark van der Wilk as well as ST John and James Hensman and was published as 'Learning Invariances using the Marginal Likelihood' (van der Wilk et al., 2018). My main contributions were the differentiable parameterisation of the invariances as well as the joint development of the variational inference scheme.

4.1 PROBLEM STATEMENT

In supervised learning, we want to predict some quantity $y \in \mathcal{Y}$ given an input $\mathbf{x} \in \mathcal{X}$ from a limited number of N training examples $\{\mathbf{x}_n, y_n\}_{n=1}^N$. We want our model to make correct predictions in as much of the input space $\mathcal{X}$ as possible. By constraining the predictor to make similar predictions for inputs which are modified in ways that are irrelevant to the prediction – for example, small translations, rotations, or deformations in the case of handwritten digits – we can generalise what we learn from a single training example to a wide range of new inputs. Invariances are invariably linked to symmetries, a fact which is referred to in physics as *Noether's theorem* (Noether, 1918). It is common to encourage a model to respect these *invariances* by training on a dataset that is enlarged by training examples that have undergone modifications that are known to not influence the output – a technique known as *data augmentation*. Developing an augmentation for a particular dataset relies on expert knowledge, trial and error, and cross-validation. This human input makes data augmentation undesirable from a machine learning perspective, akin to hand-crafting features. It is also unsatisfactory from a Bayesian perspective, according to which assumptions and expert knowledge should only be explicitly encoded in the prior distribution. By adding data that are not true observations, the posterior may become overconfident, and the marginal likelihood can no longer be used for model comparison.

data augmentation

Here, we therefore argue that data augmentation should be formulated as an invariance in the functions that are learnable by the model. To do so, we investigate prior distributions which incorporate invariances. The main benefit of treating invariances in this way is that models with different invariances can be compared using the marginal likelihood. As a consequence, differentiably parameterised invariances can even be learned by backpropagating through the marginal likelihood.

We focus on Gaussian process (GP) models for two reasons: (i) high-quality approximations for the marginal likelihood are available as discussed in Chapters 2 and 3, and (ii) the properties of a Gaussian process prior and the sample functions it can represent are fully determined by its covariance function; all symmetries and related inductive biases we want to impose on draws from the GP prior can be encoded in the covariance function. Consequently, we propose a new covariance function for GPs that can encode arbitrary discrete and continuous symmetries. Our approach overcomes the major technical obstacle that our invariant kernels cannot be computed in closed form, which has been a requirement for kernel methods until now. Instead, we only require unbiased estimates of the kernel for learning the GP and its hyperparameters. The estimate is constructed by sampling from a distribution that characterises the invariance.

These results provide a Bayesian alternative to data augmentation, and a natural method for learning invariances in a supervised manner. Additionally, the ability to use kernels that do not admit closed-form evaluation may be of use for the kernel community in general, as it may open the door to new kernels with interesting properties beyond invariance.

4.2 PREVIOUS APPROACHES TO INCORPORATING INVARIANCES

Incorporating invariances into machine learning models is commonplace, and has been addressed in many ways over the years. Despite the wide variety of methods for incorporating given invariances, there are few solutions for learning which invariances to use. Our approach is unique in that it performs direct end-to-end training using a supervised objective function. Here we present a brief review of existing methods, grouped into three rough categories.

4.2.1 *Data augmentation*

As discussed, data augmentation refers to creating additional training examples by transforming training inputs in ways that do not change the prediction (Beymer and Poggio, 1995; Niyogi et al., 1998). The enlarged dataset constrains the model's predictions to be correct for a larger region of the input space. For example, Loosli et al. (2007), building on work by Simard et al. (2000), propose augmenting the infiniteMNIST dataset with small rotations, scaling, thickening/thinning and deformations. Schölkopf et al. (1996) incorporate invariances in support vector machines by applying transformation to support vectors to obtain *virtual support vectors*; Lawrence et al. (2005) apply the same idea to the sparse representations of the information vector machine to obtain *virtual informative vectors*. On modern deep learning tasks like ImageNet (Deng et al., 2009), it is standard to apply flips, crops, and colour alterations (Krizhevsky et al., 2012; He et al., 2016). Most attempts at learning the data augmentation focus on generating more data from unsupervised models trained on the inputs. Hauberg et al. (2016) learn a distribution of mappings that transform between pairs of input images, which are then sampled and applied to random training images, while Antoniou et al. (2017) use a GAN to capture the input density.

4.2.2 *Model constraints*

An alternative to letting a flexible model learn an invariance described by a data augmentation is to constrain the model to exhibit invariances through clever parameterisation. Convolutional neural networks (CNNs) (LeCun et al., 1989;

LeCun et al., 1998) are a ubiquitous example of this, and work by applying the same filters across different image locations, giving a form of translation invariance. Cohen and Welling (2016) extend this with filters that are shared across other transformations like rotations. Invariances have also been incorporated into kernel methods. MacKay (1998) introduced the periodic kernel for functions invariant to shifts by the period. More sophisticated invariances suitable for images, like translation invariance, were discussed by Kondor (2008). Ginsbourger et al. (2012, 2013, 2016) investigated similar kernels in the context of Gaussian processes. More recently, van der Wilk et al. (2017) introduced a Gaussian process with generalisation properties similar to CNNs, together with a tractable approximation. The same method can also be used to improve the scaling of the invariant kernels introduced by the earlier papers, and our method is based on it. For similar kernels, Raj et al. (2017) present a random feature based model approximation for invariances that are not learned. A final approach is to map the inputs to some fundamental space which is constant for all inputs that we want to be invariant to (Kondor, 2008; Ginsbourger et al., 2012). For example, we can achieve rotational invariance by mapping the input image to some canonical rotation, on which classification is then performed. Jaderberg et al. (2015) do this by learning to "untransform" digits to a canonical orientation before performing classification.

4.2.3 *Regularisation*

Instead of placing hard constraints on the functions that can be represented, regularisation encourages desired solutions by adding extra penalties to the objective function. Simard et al. (1992) encourage locally invariant solutions by penalising the derivative of the classifier in the directions that the model should be invariant to. SVMs have also been regularised to encourage invariance to local perturbations, notably in Schölkopf et al. (1998), Chapelle and Schölkopf (2002), and Graepel and Herbrich (2004).

4.3 THE INFLUENCE OF INVARIANCE ON THE MARGINAL LIKELIHOOD

We aim to improve the generalisation ability of a function $f : \mathcal{X} \to \mathcal{Y}$ by constraining it to be invariant. By following the Bayesian approach and making the invariance part of the prior on $f(\cdot)$, we can use the marginal likelihood to learn the correct invariances in a supervised manner by Bayesian model selection (see Section 1.2). In this section we first define what we mean by invariance and then illustrate on a simple example how the marginal likelihood, rather than the

regular likelihood, serves as a natural objective. This illustration is similar in spirit to the polynomial regression example discussed in Section 1.2.

4.3.1 *Invariance*

We distinguish between what we refer to as "strict invariance" and "insensitivity". For the definition of strict invariance we follow the standard definition that is also used by Kondor (2008, Section 4.4) and Ginsbourger et al. (2012), where we require the value of the function $f(\cdot)$ to remain unchanged if any transformation $t : \mathcal{X} \to \mathcal{X}$ from a set $\mathcal{T}$ is applied to the input:

invariance

$$f(\mathbf{x}) = f(t(\mathbf{x})) \qquad \forall \mathbf{x} \in \mathcal{X} \qquad \forall t \in \mathcal{T}\,. \tag{4.1}$$

The set of all transformations $\mathcal{T}$ determines the invariance. For example, $\mathcal{T}$ would be the set of all rotations if we want $f(\cdot)$ to be rotationally invariant.

For many tasks, imposing strict invariance is too restrictive. For example, imposing rotational invariance will likely help the recognition of handwritten 2s, especially if they are presented in a haphazardly rotated way, while this same invariance may be detrimental for telling apart 6s and 9s in their natural orientation. For this reason, our main focus is on approximate invariances, where we want our function to not change "too much" after transformations on the input. We call this notion of invariance insensitivity. This notion is actually the most common in the related work, with data augmentation and regularisation approaches only enforcing $f(\cdot)$ to take similar values for transformed inputs, rather than exactly the same value.

We formalise insensitivity as controlling the probability of a large deviation in $f(\cdot)$ after applying a random transformation $t \in \mathcal{T}$ to the input:

insensitivity

$$P\left([f(\mathbf{x}) - f(t(\mathbf{x}))]^2 > L\right) < \epsilon \qquad \forall \mathbf{x} \in \mathcal{X} \qquad t \sim p(t)\,, \tag{4.2}$$

where $p(t)$ is a distribution over possible transformations. When working with insensitivity in practice, we conceptually think about the distribution of points that are generated by the transformations, rather than the transformations themselves. This gives a formulation that is much closer to the aim of data augmentation: a limit on the variation of the function for augmented points. Writing the distribution of points $\mathbf{x}_a$ obtained from applying the transformations to an input $\mathbf{x}$ as $p(\mathbf{x}_a \mid \mathbf{x})$, we can instead write:

$$P\left([f(\mathbf{x}) - f(\mathbf{x}_a)]^2 > L\right) < \epsilon \qquad \forall \mathbf{x} \in \mathcal{X} \qquad \mathbf{x}_a \sim p(\mathbf{x}_a \mid \mathbf{x})\,. \tag{4.3}$$

For the remainder of this chapter we refer to both strict invariance and insensitivity simply as "invariance". Our method treats both similarly, with strict invariance being a special case of insensitivity.

From these definitions, we can see how these constraints can improve generalisation. While the prediction of a non-invariant learning method is only influenced in a small region around a training point, invariant models are constrained to make similar predictions in a much larger area, with the area being determined by the set or distribution of transformations. Insensitivity is particularly useful, as it allows local invariances. Making $f(\cdot)$ insensitive to rotation can help the classification of 6s that have been rotated by small angles, while also allowing $f(\cdot)$ to give a different prediction for 9s.

4.3.2 *Invariances and the marginal likelihood*

In Section 1.2 we discussed that the marginal likelihood is a sensible objective function for probabilistic models as it incorporates a Bayesian version of Occam's razor as inductive bias for model selection. Similar to the example of polynomial regression there, we here present a second illustrative example of a simple dataset that is invariant to switching the two input coordinates, (Figure 4.1 *(left)*). We can train a model with the invariance embedded into the prior (Figure 4.1 *(middle)*), and a non-invariant model (Figure 4.1 *(right)*). In terms of RMSE on the training set (which is proportional to the log likelihood), both models fit the training data very well, with the non-invariant model even fitting slightly better. However, the invariant model generalises much better as illustrated by the test RMSE, because points on one half of the input inform the function on the other half.

Like in the previous chapters, the marginal likelihood is found by integrating the likelihood $p(\mathbf{y} \mid f)$ over the prior on $f(\cdot)$,

$$p(\mathbf{y} \mid \theta) = \int p(\mathbf{y} \mid f) p(f \mid \theta) \, \mathrm{d}f \,, \tag{4.4}$$

where θ denotes the hyperparmeters. Our example in Figure 4.1 shows that the marginal likelihood does correctly identify the invariant model as the one that generalises best.

To understand how the invariance affects the marginal likelihood, and why a high marginal likelihood can indicate good generalisation performance, we

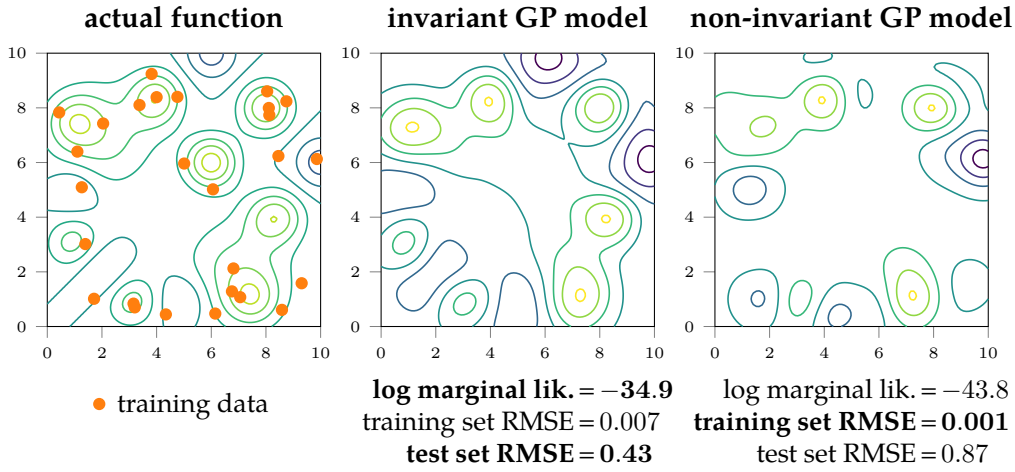


Figure 4.1: Data from a symmetric function *(left)* with the solutions learned by invariant *(middle)* and non-invariant *(right)* Gaussian processes. While the non-invariant model fits better to the training data, the invariant model generalises better. The marginal likelihood identifies the best model.

decompose it using the product rule of probability and by splitting up our dataset $\mathbf{y}$ into chunks $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_C\}$:

$$p(\mathbf{y} \mid \theta) = p(\mathbf{y}_1 \mid \theta) p(\mathbf{y}_2 \mid \mathbf{y}_1, \theta) p(\mathbf{y}_3 \mid \mathbf{y}_{1:2}, \theta) \prod_{c=4}^{C} p(\mathbf{y}_c \mid \mathbf{y}_{1:c-1}, \theta)\,. \tag{4.5}$$

From this we see that the marginal likelihood measures how well previous chunks of data predict future ones. It seems reasonable that if previous chunks of the training set accurately predict later ones, that our entire training set will predict well on a test set as well. We can apply this insight to the example in Figure 4.1 by dividing the training set into chunks consisting of the points in the top left, and the bottom right halves. The non-invariant model only generalises locally, and will therefore make predictions close to the prior for the opposing half. The invariant model is constrained to predict exactly the same for the opposing half as it has learned for the observed half, and will therefore be confident *and* correct, giving a much higher marginal likelihood.

Alternatively, we can again use a volume argument to understand the superior marginal likelihood of the invariant model in this case (see Section 1.2): The invariant model is restricted to symmetric functions and therefore incurs a smaller complexity penalty compared to the non-invariant model, which can represent a larger class of functions; at the same time, the data is fitted similarly well by both models.

Note that if the invariance had been detrimental to predictive performance (e.g. if $f(\cdot)$ was actually anti-symmetric) the marginal likelihood would have been poor, as the invariant model would have made incorrect predictions for $\mathbf{y}_2$.

Given that the marginal likelihood correctly identifies which invariances in the prior benefit generalisation, we focus our efforts in the rest of this chapter on finding a good approximation that can be practically used to learn invariances.

4.4 INFERENCE FOR GAUSSIAN PROCESSES WITH INVARIANCES

Marginal likelihoods are commonly hard to compute, but good approximations exist for Gaussian processes with simple kernels as we have seen in Chapter 2. In this section, we focus our efforts on Gaussian processes based on kernels with complex, parameterised invariances built in, for which we derive a practical marginal likelihood approximation.

Our approximation is based on the variational lower bounds for Gaussian processes, which we introduced in Sections 2.4 and 2.5. While Turner and Sahani (2011) point out that variational bounds do introduce bias to hyperparameter estimates, the bias is well-understood in our case, and is reduced by using sufficiently many inducing points as we discussed in Chapter 3.

4.4.1 *Invariant and insensitive Gaussian processes*

Our starting point is the double-sum construction for priors over strictly invariant functions (Kondor, 2008; Ginsbourger et al., 2012), which we briefly review. If $f(\cdot)$ is strictly invariant to a set of transformations, $f(\cdot)$ must also be invariant to compositions of transformations, as the same invariance holds at the transformed point $t(\mathbf{x})$. The set of all such compositions of transformations forms a *group* G. We refer to the set of all points that can be obtained by applying transformations in G to a point $\mathbf{x}$ as the *orbit* of $\mathbf{x}$ and denote it by $\mathcal{A}(\mathbf{x}) = \{t(\mathbf{x}) \mid t \in G\}$. We use A to denote the number of elements in the orbit $A := |\mathcal{A}(\mathbf{x})|$; for continuous symmetries, A can be infinite. We first focus on discrete symmetries and address continuous symmetries below. All input points which can be transformed into one another by an element of G share an orbit, and must also have the same function value. This implies a simple construction of a strictly invariant function $f(\cdot)$ from a non-invariant function $g(\cdot)$: we simply sum $g(\cdot)$ over the orbit of a point,

orbit

$$f(\mathbf{x}) = \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} g(\mathbf{x}_a)\,. \tag{4.6}$$

By placing a GP prior on $g(\cdot)$, $g(\cdot) \sim \mathcal{GP}\left(0, k_g(\cdot,\cdot)\right)$, we imply a GP on invariant functions $f(\cdot)$, because Gaussians are closed under summation; the sum of two

Gaussian random variables is again a Gaussian random variable. We find that the induced prior on $f(\cdot)$ has a double-sum kernel:

$$k_f(\mathbf{x}, \mathbf{x}') = \mathbb{E}_g \left[\sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} g(\mathbf{x}_a) \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x}')} g\left(\mathbf{x}'_a\right) \right] \tag{4.7}$$

$$= \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x}')} \mathbb{E}_g \left[g(\mathbf{x}_a) g\left(\mathbf{x}'_a\right) \right] \tag{4.8}$$

$$= \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x}')} k_g\left(\mathbf{x}_a, \mathbf{x}'_a\right) . \tag{4.9}$$

Earlier we argued that insensitivity is often more desirable. In order to relax the constraint of strict invariance, we relax the constraint that we sum over an orbit. Therefore, we consider $\mathcal{A}(\mathbf{x})$ to be an arbitrary set of points, which we refer to as an *augmentation set*, describing what input changes $f(\cdot)$ should be insensitive to. If two inputs have significantly overlapping augmentation sets, then their corresponding function values are constrained to be similar, as many terms in the sum of Equation (4.6) are shared. This kernel was independently also studied by Dao et al. (2019) as a first-order approximation of data augmentation, and Raj et al. (2017) as a local invariance.

augmentation set

We can also consider infinite augmentation sets, where we describe the relative density of elements using a probability density, which we refer to as the *augmentation density* $p(\mathbf{x}_a \mid \mathbf{x})$. We can think of $p(\mathbf{x}_a \mid \mathbf{x})$ as a process which perturbs the training data, much like how data augmentation is performed. Following a similar argument to the above, this results in a kernel that is doubly integrated over the augmentation distribution $p(\mathbf{x}_a \mid \mathbf{x})$:

augmentation distribution

$$k_f(\mathbf{x}, \mathbf{x}') = \iint p\left(\mathbf{x}_a \mid \mathbf{x}\right) p\left(\mathbf{x}'_a \mid \mathbf{x}'\right) k_g(\mathbf{x}_a, \mathbf{x}'_a) \, \mathrm{d}\mathbf{x}_a \, \mathrm{d}\mathbf{x}'_a \, . \tag{4.10}$$

The set of hyperparameters θ of the kernel k_f consist of the parameters of the augmentation distribution $p(\mathbf{x}_a \mid \mathbf{x})$ and the base kernel k_g. We treat them as hyperparameters of the model and tune them using an approximation to the marginal likelihood. For brevity we have dropped θ from the notation.

When using these kernels, we face an additional obstacle on top of the usual problems with scalability and non-conjugate inference. The sums over large orbits prohibit the evaluation of Equation (4.9), while the integrals in Equation (4.10) are analytically intractable for interesting invariances. Starting from Section 4.4.3, we develop an approximation that allows us to evaluate a lower bound to the marginal likelihood which only requires samples of the orbit $\mathcal{A}(\mathbf{x})$ or augmentation distribution $p(\mathbf{x}_a \mid \mathbf{x})$. This allows us to minibatch not only over examples in the training set, but also over samples that describe the desired invariances.

However, before we present the inference scheme, we explain how the presented covariance function relates to our above notion of insensitivity.

4.4.2 *Insensitivity*

Above, we described that we want to construct functions that are insensitive to certain variations of the inputs. We defined the possible variations for a particular input $\mathbf{x}$ through the augmentation distribution $p(\mathbf{x}_a \mid \mathbf{x})$ and used it construct an invariant or insensitive double sum kernel. In the following, we discuss the connections to the notion of insensitivity introduced in Section 4.3.1, which we restate here:

$$P\left([f(\mathbf{x}) - f(\mathbf{x}_a)]^2 > L\right) < \epsilon \qquad \forall \mathbf{x} \in \mathcal{X} \qquad \mathbf{x}_a \sim p(\mathbf{x}_a \mid \mathbf{x})\,. \tag{4.3}$$

For a given ϵ, a smaller L implies more insensitivity. Here, we aim to quantify the degree of insensitivity for priors with double sum kernels.

To simplify our analysis we assume that our kernel is defined as

$$k(\mathbf{x}, \mathbf{x}') = \frac{\sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x}')} k_g(\mathbf{x}_a, \mathbf{x}'_a)}{\sqrt{\sum_{\mathbf{x}_a, \mathbf{x}_b \in \mathcal{A}(\mathbf{x})} k_g(\mathbf{x}_a, \mathbf{x}_b)} \sqrt{\sum_{\mathbf{x}'_a, \mathbf{x}'_b \in \mathcal{A}(\mathbf{x}')} k_g(\mathbf{x}'_a, \mathbf{x}'_b)}}\,, \tag{4.11}$$

that is, we normalise the double sum kernel. We do this to ensure that we always retain a unit marginal variance $k(\mathbf{x}, \mathbf{x}')$, in order to ensure that our kernel can actually learn something. Without this constraint, we can trivially pick $\mathcal{A}(\mathbf{x})$ to average many distant points, which makes all $f(\cdot)$ insensitive, but also completely constant. We do not need to apply this constraint in our practical method, as we choose the scale of the kernel by optimising the marginal likelihood.

We start by bounding the deviation of functions under the prior between $\mathbf{x}$ and $\mathbf{x}_a$.

Lemma 4.1
When $f \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ we can bound the probability of a deviation as

$$P\left([f(\mathbf{x}) - f(\mathbf{x}_a)]^2 > 2\epsilon(1 - \mathbb{E}_{\mathbf{x}_a|\mathbf{x}} k(\mathbf{x}, \mathbf{x}_a))\right) < \frac{1}{\epsilon}\,. \tag{4.12}$$

Proof.
We apply Markov's inequality to the random variable $[f(\mathbf{x}) - f(\mathbf{x}_a)]^2$:

$$P\left([f(\mathbf{x}) - f(\mathbf{x}_a)]^2 > \epsilon \mathbb{E}_{f, \mathbf{x}_a|\mathbf{x}} [f(\mathbf{x}) - f(\mathbf{x}_a)]^2\right) < \frac{1}{\epsilon}\,. \tag{4.13}$$

The expectation over $f(\cdot)$ evaluates as

$$\begin{aligned}\mathbb{E}_f[f(\mathbf{x}) - f(\mathbf{x}_a)]^2 &= \mathbb{E}_f\left[f(\mathbf{x})^2 - 2f(\mathbf{x})f(\mathbf{x}_a) + f(\mathbf{x}_a)^2\right] \\ &= k(\mathbf{x},\mathbf{x}) - 2k(\mathbf{x},\mathbf{x}_a) + k(\mathbf{x}_a,\mathbf{x}_a) \\ &= 2\left(1 - k(\mathbf{x},\mathbf{x}_a)\right)\,, \end{aligned} \tag{4.14}$$

leaving only the expectation over $\mathbf{x}_a$ as in the statement. □

This shows that we can increase the insensitivity of functions in the prior by increasing $\mathbb{E}_{\mathbf{x}_a|\mathbf{x}}k(\mathbf{x},\mathbf{x}_a)$. Not all distributions $p(\mathbf{x}_a \mid \mathbf{x})$ actually increase the expected covariance. In our method, we simply parameterise kernels that allow insensitivity, and then let the marginal likelihood determine the extent to which it is applied. As an example, we take $k_g(\mathbf{x},\mathbf{x}')$ to be a squared exponential kernel. We can increase the expected covariance by ensuring that the augmentation densities of $\mathbf{x}$ and a random augmented point $\mathbf{x}_a$ overlap largely. If the augmentation distributions fully and uniformly overlap, we obtain strict invariance again.

4.4.3 *Variational inference using inducing points*

We want to use the invariant GP defined in the previous section as a prior over functions for regression and classification models.

Notation

We follow the same notation as in Chapters 2 and 3 and denote the prior as $p(f)$, sets of inputs as matrices $\mathbf{X} \in \mathbb{R}^{N\times D}$, and observations and function values as vectors, $\mathbf{y} = \{y_n\}_{n=1}^N$ and $\mathbf{f} = \{f(\mathbf{x}_n)\}_{n=1}^N = f(\mathbf{X})$, respectively. We denote our model:

$$\begin{aligned} f \mid \theta &\sim \mathcal{GP}\left(0, k_f(\cdot,\cdot)\right) \\ y_n \mid f, \mathbf{x}_n &\overset{\text{i.i.d}}{\sim} p(y_n \mid f(\mathbf{x}_n))\,, \end{aligned} \tag{4.15}$$

Stochastic variational objective function for Gaussian processes

As we have already discussed in Chapters 1 and 2 inference in GPs suffers from two main difficulties. First, inference is only analytically tractable for Gaussian likelihoods. Second, computation in GP models is well-known to scale badly with the dataset size, requiring $\mathcal{O}(N^3)$ computations and $\mathcal{O}(N^2)$ memory on $K_{\mathbf{ff}}$.

We therefore use the variational sparse GP approximation methods for general likelihoods that support stochastic optimisation (Titsias, 2009a; Hensman et al.,

2013, 2015), refer to Sections 2.4 and 2.5 for an introduction. We start from the ELBO in Equation (2.29), which we restate here for convenience:

$$\log p(\mathbf{y}) \geq \mathcal{F} = \sum_{n=1}^{N} \mathbb{E}_{q(f(\mathbf{x}_n))} \log p(\mathbf{y}|f(\mathbf{x}_n) - \mathrm{KL}(\phi(\mathbf{u}) \parallel p(\mathbf{u}))\,. \tag{4.16}$$

We find the expectation under $q(f(\mathbf{x}_n))$ either analytically or by Monte Carlo. In order to reduce the cost of evaluating the whole sum, we evaluate the bound stochastically on minibatches of $\widetilde{N}$ datapoints. This technique allows Gaussian processes to be applied to large datasets with general likelihoods. However, it does not address the issue of having to evaluate the intractable double sum kernel k_f in Equations (4.9) and (4.10).

4.4.4 *Inter-domain inducing variables*

The problem of evaluating large double sums was encountered by van der Wilk et al. (2017) for convolutional and strictly invariant kernels. Their solution is based on the observation that problems with evaluating the bound in Equation (4.16) stem from intractabilities in the approximate posterior $q(\mathbf{f})$, because this is where the kernel evaluations are needed. They show that by choosing a different parameterisation of $q(f)$, the cost of evaluating the approximate posterior for a minibatch of $\widetilde{N}$ points can be reduced from $\mathcal{O}(\widetilde{N}^2 + (\widetilde{N}M + M^2)A^2 + M^3)$ to $\mathcal{O}(\widetilde{N}A^2 + \widetilde{N}MA + M^2 + M^3)$ – a significant saving, particularly when $\widetilde{N}$ is small, and M and A are large. Here, $\widetilde{N}$ denotes the minibatch size, A the orbit size, and M the number of inducing points.

This can be achieved simply by constructing the posterior from inducing variables in $g(\cdot)$ instead of $f(\cdot)$. Approximations constructed using observations in different spaces are said to use *inter-domain* inducing variables (Figueiras-Vidal and Lázaro-Gredilla, 2009), and can use the same variational bound as in Equation (4.16) (Matthews et al., 2016), with only $K_{\mathbf{uu}}$ and $\mathbf{k}_{\mathbf{u}}(\cdot)$ being modified in $q(f(\mathbf{x}_n))$:

$$\begin{aligned} k_{\mathbf{fu}}(\mathbf{x}, \mathbf{z}) &= \mathbb{E}_{p(g)}[f(\mathbf{x})g(\mathbf{z})] = \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} k_g(\mathbf{x}_a, \mathbf{z}) \\ k_{\mathbf{uu}}(\mathbf{z}, \mathbf{z}') &= k_g(\mathbf{z}, \mathbf{z}')\,. \end{aligned} \tag{4.17}$$

The new covariances require only a single sum, or no sum at all. Only $k_f(\mathbf{x}_n, \mathbf{x}_n)$ still requires a double sum, although this can be mitigated by keeping the minibatch size $\widetilde{N}$ small.

While this technique allows variational inference to be applied to invariant kernels with moderately sized orbits, similar to the convolutional kernels in van

der Wilk et al. (2017), it does not help when even a single sum is too large. This approach also does not apply when intractable integrals appear as is the case in Equation (4.10), because evaluations of the intractable k_f are still needed, and the covariance function $k_{\mathbf{fu}}$ also requires an intractable integral:

$$k_{\mathbf{fu}}(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{p(g)} \int p(\mathbf{x}_a \mid \mathbf{x}) g(\mathbf{x}_a) g(\mathbf{z}) \, \mathrm{d}\mathbf{x}_a = \int p(\mathbf{x}_a \mid \mathbf{x}) k_g(\mathbf{x}_a, \mathbf{x}) \, \mathrm{d}\mathbf{x}_a \,. \quad (4.18)$$

4.4.5 *An unbiased estimator using samples describing invariances*

We now show that the inter-domain parameterisation above allows us to create an unbiased estimator of the lower bound in Equation (4.16) using unbiased Monte Carlo estimates of k_f and $k_{\mathbf{fu}}$. That is, we approximate them by evaluating them on a subsampled minibatch from the augmentation set or the augmentation distribution, respectively. Note that this derivation only holds for Gaussian likelihoods. Some non-Gaussian likelihoods can be rewritten as Gaussian likelihoods and we discuss them in the next Section 4.4.6.

For Gaussian likelihoods, the ELBO in Equation (4.16) can be evaluated analytically as

$$\mathcal{L} = \sum_{n=1}^{N} \left[-\log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \left(y_n^2 - 2 y_n \mu_n + \mu_n^2 + \sigma_n^2 \right) \right] - \mathrm{KL}(q(\mathbf{u}) \parallel p(\mathbf{u})) \,. \quad (4.19)$$

In this expression, $\mu_n = \mu(\mathbf{x}_n)$ and $\sigma_n^2 = \nu(\mathbf{x}_n, \mathbf{x}_n)$ (Equation (2.24)) are the only terms which depend on the intractable covariances k_f and $k_{\mathbf{fu}}$. The KL term is tractable, as it only depends on $K_{\mathbf{uu}}$, which can be evaluated from k_g directly, see Equation (4.17).

In the following, we construct unbiased estimators $\widehat{\mu_n}$, $\widehat{\mu_n^2}$ and $\widehat{\sigma_n^2}$ for the intractable terms, which only rely on samples of $p(\mathbf{x}_a \mid \mathbf{x})$.

The posterior mean can be estimated easily using a Monte Carlo estimate of $k_{\mathbf{fu}}$:

$$\mu_n = \mathbf{k}_{\mathbf{f}_n \mathbf{u}} K_{\mathbf{uu}}^{-1} \mathbf{m} = \left[\int p(\mathbf{x}_a | \mathbf{x}_n) \mathbf{k}_g(\mathbf{x}_a, \mathbf{Z}) \, \mathrm{d}\mathbf{x}_a \right] K_{\mathbf{uu}}^{-1} \mathbf{m} \quad (4.20)$$

$$\implies \widehat{\mu_n} = \widehat{\mathbf{k}}_{\mathbf{f}_n \mathbf{u}} K_{\mathbf{uu}}^{-1} \mathbf{m} \,, \quad (4.21)$$

$$\widehat{k}_{\mathbf{fu}}(\mathbf{x}, \mathbf{z}) = \sum_{s=1}^{S} k_g(\mathbf{x}^{(s)}, \mathbf{z}), \qquad \mathbf{x}^{(s)} \sim p(\mathbf{x}_a \mid \mathbf{x}). \quad (4.22)$$

For μ_n^2 and σ_n^2, we start by rewriting them so we can focus on estimators of $k_f(\mathbf{x}_n, \mathbf{x}_n)$ and $\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\intercal \mathbf{k}_{\mathbf{f}_n\mathbf{u}}$:

$$\mu_n^2 = \mathbf{k}_{\mathbf{f}_n\mathbf{u}} K_{\mathbf{uu}}^{-1} \mathbf{m}\mathbf{m}^\intercal K_{\mathbf{uu}}^{-1} \mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\intercal = \operatorname{tr}\left[K_{\mathbf{uu}}^{-1} \mathbf{m}\mathbf{m}^\intercal K_{\mathbf{uu}}^{-1} \left(\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\intercal \mathbf{k}_{\mathbf{f}_n\mathbf{u}}\right)\right] , \tag{4.23}$$

$$\sigma_n^2 = k_f(\mathbf{x}_n, \mathbf{x}_n) - \operatorname{tr}\left[K_{\mathbf{uu}}^{-1} \left(K_{\mathbf{uu}} - \mathbf{S}\right) K_{\mathbf{uu}}^{-1} \left(\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\intercal \mathbf{k}_{\mathbf{f}_n\mathbf{u}}\right)\right] . \tag{4.24}$$

We treat $k_f(\mathbf{x}_n, \mathbf{x}_n)$ and an element of $\mathbf{k}_{\mathbf{f}_n\mathbf{u}}^\intercal \mathbf{k}_{\mathbf{f}_n\mathbf{u}}$ identically, as they can both be written as the following double integral or double sum

$$\begin{aligned} I &= \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x})} r(\mathbf{x}_a, \mathbf{x}'_a) \,, \\ I &= \iint p(\mathbf{x}_a \mid \mathbf{x}_n) p(\mathbf{x}'_a \mid \mathbf{x}_n) r(\mathbf{x}_a, \mathbf{x}'_a) \, \mathrm{d}\mathbf{x}_a \, \mathrm{d}\mathbf{x}'_a \end{aligned} \tag{4.25}$$

with $r(\mathbf{x}_a, \mathbf{x}'_a) = k_g(\mathbf{x}_a, \mathbf{x}'_a)$ and $r = k_{\mathbf{fu}}(\mathbf{x}_a, \mathbf{z}_m) k_{\mathbf{fu}}(\mathbf{x}'_a, \mathbf{z}_{m'})$, respectively.

A simple Monte Carlo estimate of I would require sampling two independent minibatches of points for $\mathbf{x}_a$ and $\mathbf{x}'_a$. However, given the cost of transforming input images, we aim to use the *same* subset for both sums. This additionally speeds up the kernel computations, as the same covariances with the inducing points are needed.

We propose to use the following estimators that only use a single minibatch $\mathcal{S}$ of augmentation points with size $S := |\mathcal{S}|$. For the double sum kernel it is given by

$$\widehat{I}_{\sum\sum} = \frac{A(A-1)}{S(S-1)} \sum_{s=1}^{S} \sum_{\substack{s'=1 \\ s' \neq s}}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) + \frac{A}{S} \sum_{s=1}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s)}) \,, \tag{4.26}$$

where the summation is over points $\mathbf{x}^{(s)}$ from a minibatch of augmentation points $\mathcal{S} \subset \mathcal{A}(\mathbf{x})$ that has been sampled *without* replacement from $\mathcal{A}(\mathbf{x})$, where $A := |\mathcal{A}(\mathbf{x})|$. Note that we have to reweight the diagonal and off-diagonal terms in the sums because using the same minibatch increases the relative importance of the diagonal terms. We discuss this further in the proof of Proposition 4.1. Similarly, we propose the following estimator for the double integral kernel:

$$\widehat{I}_{\int\int} = \frac{1}{S(S-1)} \sum_{s=1}^{S} \sum_{s'=1}^{S} r\left(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}\right) (1 - \delta_{ss'}) \,, \qquad \mathbf{x}^{(s)} \sim p(\mathbf{x}_a \mid \mathbf{x}) \,, \tag{4.27}$$

where the summation is over points $\mathbf{x}^{(s)}$ from a minibatch of augmentation points $\mathcal{S}$ whose elements have been drawn i.i.d. from $p(\mathbf{x}_a \mid \mathbf{x})$. Given these unbiased estimates, we can evaluate and optimise the variational lower bound.

Before we discuss non-Gaussian likelihoods in Section 4.4.6 and move on to experiments, we briefly discuss these estimators and proof that they are unbiased in Propositions 4.1 and 4.2.

Proposition 4.1 (Unbiasedness of the estimator $\widehat{I}_{\sum\sum}$)
The estimator $\widehat{I}_{\sum\sum}$ in Equation (4.26) is unbiased.

Proof.
The original double sum in Equation (4.25) can be decomposed into diagonal and off-diagonal terms

$$I = \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x})} r(\mathbf{x}_a, \mathbf{x}'_a) \tag{4.25}$$

$$= \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} \sum_{\substack{\mathbf{x}'_a \in \mathcal{A}(\mathbf{x}) \\ \mathbf{x}'_a \neq \mathbf{x}_a}} r(\mathbf{x}_a, \mathbf{x}'_a) + \sum_{\mathbf{x}_a \in \mathcal{A}(\mathbf{x})} r(\mathbf{x}_a, \mathbf{x}_a) \tag{4.28}$$

$$= I_{\neg d} + I_d \,. \tag{4.29}$$

The size of $\mathcal{A}$ is given by $A = |\mathcal{A}|$ and there are $A(A-1)$ off-diagonal and A diagonal terms in this sum. We now subsample one minibatch $\mathcal{S} \subset \mathcal{A}$ of size $S = |\mathcal{S}|$ without replacement from $\mathcal{A}$.

A naive estimator would be given by:

$$\widehat{I}_{\text{naive}} = \sum_{s=1}^{S} \sum_{s'=1}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) \tag{4.30}$$

However, this estimator has $S(S-1)$ off-diagonal and S diagonal terms, which is different from the relative frequencies for the original double sum above, such that the $\widehat{I}_{\text{naive}}$ cannot be unbiased. However, we can unbias it by reweighting the relative frequencies as was done for the estimator $\widehat{I}_{\sum\sum}$ in Equation (4.26). More formally, we need to show that

$$\mathbb{E}_{p(\mathcal{S})} \widehat{I}_{\sum\sum} \stackrel{!}{=} I_{\sum\sum} \tag{4.31}$$

where $p(\mathcal{S})$ denotes the sampling of the minibatch. The probability that a particular index s gets drawn is $\frac{1}{A}$. Similarly, because we sample without replacement, the probability of two indices s and s' with $s' \neq s$ to get drawn is $\frac{1}{A(A-1)}$. We summarise this as

$$p(\mathbf{x}_s = \mathbf{x}_a, \mathbf{x}_{s'} = \mathbf{x}'_a) = (1 - \delta(\mathbf{x}_a - \mathbf{x}'_a))(1 - \delta_{ij}) \tfrac{1}{A(A-1)} + \delta_{ij} \delta(\mathbf{x}_a - \mathbf{x}'_a) \tfrac{1}{A} \tag{4.32}$$

We can now evaluate the above expectation:

$$\mathbb{E}_{p(\mathcal{S})}\widehat{I}_{\sum\sum} = \sum_{\substack{s=1\\s'=1}}^{S} \sum_{\substack{\mathbf{x}_a\in\mathcal{A}(\mathbf{x})\\ \mathbf{x}'_a\in\mathcal{A}(\mathbf{x})}} p(\mathbf{x}_s = \mathbf{x}_a, \mathbf{x}_{s'} = \mathbf{x}'_a) r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) w_{ss'}\,, \tag{4.33}$$

where $w_{ss'}$ denotes the reweighting factors. Therefore,

$$\begin{aligned}\mathbb{E}_{p(\mathcal{S})}\widehat{I}_{\sum\sum} &= \sum_{s=1}^{S}\sum_{s'=1}^{S}(1-\delta_{ss'})\tfrac{w_{ss'}}{A(A-1)}I_{\neg d} + \delta_{ss'}\tfrac{w_{ss'}}{A}I_d \\ &= \tfrac{\cancel{A(A-1)}}{\cancel{S(S-1)}}\tfrac{\cancel{S(S-1)}}{\cancel{A(A-1)}}I_{\neg d} + \tfrac{\cancel{A}}{\cancel{S}}\tfrac{\cancel{S}}{\cancel{A}}I_d \qquad (4.34)\\ &= I\,. \qquad (4.35)\end{aligned}$$

For ease of implementation, we re-write the estimator in terms of a full and diagonal sum over r:

$$\widehat{I}_{\sum\sum} = \frac{A(A-1)}{S(S-1)}\sum_{s=1}^{S}\sum_{s'=1}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) + \frac{A(S-A)}{S(S-1)}\sum_{i=1}^{M} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s)})\,. \tag{4.36}$$

□

Proposition 4.2 (Unbiasedness of the estimator $\widehat{I}_{\int\int}$)
The estimator $\widehat{I}_{\int\int}$ in Equation (4.27) is unbiased.

Proof.
Informally, we can think of this as the double-sum kernel, where we take a limit to an infinite augmentation set $A \to \infty$ but also divide by A^2 to normalise. If we apply this limit to Equation (4.26) we arrive at

$$\widehat{I}_{\int\int} = \frac{1}{S(S-1)}\left[\sum_{s=1}^{S}\sum_{s'=1}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) - \sum_{s=1}^{S} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s)})\right], \tag{4.37}$$

with $\mathbf{x}^{(s)}, \mathbf{x}^{(s')} \sim p(\mathbf{x}_a \mid \mathbf{x})$. This estimator is easier to treat as the diagonal terms comprise a set of measure 0 under $p(\mathbf{x}_a \mid \mathbf{x})$ (as long as it is continuous). We can show it to be unbiased by taking the expectation over all $\mathbf{x}^{(s)}$, and a small re-arrangement:

$$\begin{aligned}\mathbb{E}_{p\left(\{\mathbf{x}^{(s)}\}_{s=1}^{S}|\mathbf{x}\right)}\widehat{I}_{\int\int} &= \frac{1}{S(S-1)}\sum_{s=1}^{S}\sum_{s'=1}^{S}(1-\delta_{ss'})\mathbb{E}_{\mathbf{x}^{(s)},\mathbf{x}^{(s')}} r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')}) \\ &= \iint p(\mathbf{x}^{(s)}|\mathbf{x})p(\mathbf{x}^{(s')}|\mathbf{x}) r(\mathbf{x}^{(s)}, \mathbf{x}^{(s')})\,\mathrm{d}\mathbf{x}^{(s)}\,\mathrm{d}\mathbf{x}^{(s')} \qquad (4.38)\\ &= I \qquad (4.39)\end{aligned}$$

□

The estimator for $k_f(\mathbf{x}, \mathbf{x}')$ costs $\mathcal{O}(S^2)$, as the double sum needs to be evaluated for each element. When $r = k_{\mathbf{fu}}(\mathbf{x}_a, \mathbf{z}_m)k_{\mathbf{fu}}(\mathbf{x}'_a, \mathbf{z}_{m'})$, the estimator costs only $\mathcal{O}(S)$, as the double sum factorises over each term.

4.4.6 *Logistic classification with Pòlya-Gamma latent variables*

The estimators we developed in the previous section allowed us to estimate the bound in an unbiased way, as long as the variational expectation over the likelihood only depended on μ_n, μ_n^2, and σ_n^2. This limits the applicability of our method to likelihoods of a Gaussian form. Luckily, some likelihoods can be written as expectations over unnormalised Gaussians. For example, the logistic likelihood can be written as an expectation over a Pòlya-Gamma variable ω (Polson et al., 2013):

$$\sigma(y_n f(\mathbf{x}_n)) = (1 + \exp(-y_n f(\mathbf{x}_n)))^{-1} \tag{4.40}$$

$$= \int c\mathcal{N}\left(f(\mathbf{x}_n); \tfrac{1}{2}y_n, \omega_n^{-1}\right) p(\omega_n)\, \mathrm{d}\omega_n\,. \tag{4.41}$$

This trick was investigated by Gibbs and MacKay (2000) and recently revisited by Wenzel et al. (2018) to construct a variational lower bound to the logistic likelihood with a Gaussian form:

$$\log p(y_n|f(\mathbf{x}_n)) = \log \sigma(y_n f(\mathbf{x}_n)) \tag{4.42}$$

$$\geq \mathbb{E}_{q(f(\mathbf{x}_n))q(\omega_n)}\left[c \log \mathcal{N}\left(f(\mathbf{x}_n); \tfrac{1}{2}y_n, \omega_n^{-1}\right)\right]\,. \tag{4.43}$$

This bound for the likelihood can be used as a drop-in approximation for the exact likelihood, at the cost of adding an additional KL gap to the true marginal likelihood. For our purpose, the crucial benefit is that we again obtain a Gaussian form in the expectation over $q(f(\mathbf{x}_n))$, for which we can use the unbiased estimators we developed above. Gibbs and MacKay (2000) and Wenzel et al. (2018) go on to find the optimal parameters for $q(\omega_n)$ in closed form. We cannot use this, as the optimal parameters depend non-linearly on μ_n and σ_n. Instead, we choose to parameterise $q(\omega_n)$ as a Pòlya-Gamma distribution with its parameters given by a recognition network mapping from the corresponding input and label, following P. and Welling (2014). This method can likely be extended to the multi-class setting using the stick-breaking construction by Linderman et al. (2015).

4.5 EMPIRICAL EVALUATION

We demonstrate our approach on a series of experiments on variants of the MNIST datasets. While MNIST has been accurately solved by other methods, we intend to show that a model like an RBF GP (Radial Basis Function or squared exponential kernel), for which MNIST is challenging, can be significantly improved by learning adequate invariances and insensitivities.

4.5.1 *Augmentation distributions*

For binary classification tasks, we use the Pòlya-Gamma approximation for the logistic likelihood, while for multi-class classification, we are currently forced to use the Gaussian likelihood.

Here, we consider two classes of transformations for which we automatically learn parameters: (i) global affine transformations (scale, rotation, shear, and translation), and (ii) local deformations as manually employed in the infiniteMNIST dataset (Loosli et al., 2007). Arguably, these are two of the simplest differentiable and parametrisable transformations that can give rise to augmentation sets. We address the advantages and disadvantages of this choice as well as possible alternatives at the end of this chapter in Section 4.6. Note that we must be able to backpropagate through the transformations in order to learn their parameters.

AFFINE TRANSFORMATIONS

A 2D affine transformation is determined by six degrees of freedom, which specify scaling, global rotation, shear, and translation. There are different ways to parameterise these degrees of freedom. A general affine transformation is defined through a transformation matrix A_ϕ with six affine parameters ϕ, which map a point (x, y) in the 2D plane to a new point (x', y') through:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \phi_1 & \phi_2 & \phi_3 \\ \phi_4 & \phi_5 & \phi_6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} . \tag{4.44}$$

Note that here (and only here) x and y do not denote input and output but coordinates of a 2D coordinate system. We can think of A_ϕ being the result of the product of several simpler transformation matrices, where each matrix encodes only rotations, scalings, shears, and translations. For

example, a more interpretable parameterisation would be a rotation by an angle α and a scaling

$$A_\phi = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} . \tag{4.45}$$

To define the augmentation distribution $p(\mathbf{x}_a \mid \mathbf{x})$, we first define a distribution on these affine parameters $p_\psi(\phi)$, which itself is parametrised through parameters ψ. Then, to sample from $p(\mathbf{x}_a \mid \mathbf{x})$, we first draw $\phi \sim p_\psi(\phi)$ and then apply the corresponding transformation to the input $\mathbf{x}$ to obtain the augmented sample $\mathbf{x}_a = \mathrm{Aff}_\phi(\mathbf{x})$. Because affine transformations are differentiable with respect to the affine parameters ϕ, we can backpropagate any loss into the parameters ψ of $p_\psi(\phi)$ using the reparameterisation trick – as long as $p_\psi(\phi)$ is reparameterisable with respect to ψ.

Here, we again make the simplest possible assumption that $p_\psi(\phi)$ factorises and each factor is given by a uniform distribution with learnable bounds, that is, minimal and maximal values,

$$p_\psi(\phi) = \prod_{i=1}^{6} p_{\psi_i}(\phi_i) , \qquad p_{\psi_i}(\phi_i) = \mathrm{Unif}\left(\phi_{i,\min}, \phi_{i,\max}\right) . \tag{4.46}$$

Thus, the learnable parameters ψ are given by $\psi = \{\phi_{i,\min}, \phi_{i,\max} \mid i = 1, \ldots, 6\}$. In other words, for each affine parameter we learn the boundaries of the range in which two samples are said to be invariant or insensitive. For example, for rotations the boundary values might be $\phi_{\alpha,\min} = -15°$ and $\phi_{\alpha,\max} = 25°$, such that the augmentation set of an image would be given by all possible rotations between those angles (we have neglected the other 5 affine transformations in this example). In practice, we use the implementation of affine transformations used in spatial transformer networks (Jaderberg et al., 2015) from `https://github.com/kevinzakka/spatial-transformer-network`, which is fully differentiable with respect to the affine parameters.

LOCAL DEFORMATIONS

As a second class of transformations we consider local deformations as introduced with the infiniteMNIST dataset (Loosli et al., 2007; Simard et al., 2000). The local deformations can be decomposed into random local

deformations, local scalings, local rotations, and a thickening/thinning transformation. A transformed image $\mathbf{x}_a$ is given by

$$\mathbf{x}_a = \sigma\left(\mathbf{x} + \mathbf{f}_x \mathbf{t}_x(\mathbf{x}) + \mathbf{f}_y \mathbf{t}_y(\mathbf{x}) + \beta\sqrt{\mathbf{t}_x(\mathbf{x})^2 + \mathbf{t}_y(\mathbf{x})^2}\right) \tag{4.47}$$

$$\mathbf{f}_x = (\alpha_x \mathbf{f}_{\text{rand},x} + \alpha_{\text{rot}} \mathbf{f}_{\text{rot},x} + \alpha_{\text{scale}} \mathbf{f}_{\text{scale},x}) \tag{4.48}$$

$$\mathbf{f}_y = (\alpha_y \mathbf{f}_{\text{rand},y} + \alpha_{\text{rot}} \mathbf{f}_{\text{rot},y} + \alpha_{\text{scale}} \mathbf{f}_{\text{scale},y}) \tag{4.49}$$

Here, σ is a clipping or squashing function to ensure that $\mathbf{x}_a$ is in the same range as $\mathbf{x}$; $\mathbf{f}_{\text{rand/rot/scale}}$ are vector fields for local random deformations, local rotations and local scalings with corresponding scale factors $\alpha_{x/y}$, α_{rot} and α_{scale}; $\mathbf{t}_{x/y}(\mathbf{x})$ denotes the image gradients of an image $\mathbf{x}$ in direction x/y; and β scales a thinning/thickening transformation.

The image gradients $\mathbf{t}_{x/y}(\mathbf{x})$ can be computed by convolving the image $\mathbf{x}$ with a derivative of Gaussian filter (Simard et al., 2000). However, to increase performance, we approximate them by convolutions with Sobel filters (Kanopoulos et al., 1988) in the x/y direction, respectively. The vector fields for local rotations, $\mathbf{f}_{\text{rot},x/y}$, and scalings, $\mathbf{f}_{\text{scale},x/y}$, are fixed and can be derived as first order approximations of the corresponding affine transformations. The vector fields for random local deformations $\mathbf{f}_{\text{rand},x/y}$ are drawn i.i.d. from a diagonal standard Normal distribution and then correlated spatially by a Gaussian smoothing kernel of width σ_d.

Similar to the case of affine transformations, we place factorised uniform distributions on the scale parameters α_* and β and learn their ranges together with the scale parameter σ_d of the random local deformations.

4.5.2 *Recovering invariances in binary MNIST classification*

As a first test, we demonstrate that our approach can recover the parameter of a known transformation in an odds-vs-even MNIST binary classification problem using the Pòlya-Gamma likelihood from Section 4.4.6. We consider the regular MNIST dataset and rotate each example by a randomly chosen angle $\phi \in [-\alpha_{\text{true}}, \alpha_{\text{true}}]$ for $\alpha_{\text{true}} \in \{90°, 180°\}$.

To create samples from the orbit $\mathcal{A}(\mathbf{x})$, we limit ourselves to rotations, that is, we transform the training examples by affine transformations that are limited to rotations, that is

$$A_\phi = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \end{pmatrix} \qquad \phi \sim \text{Uniform}([-\alpha, \alpha]). \tag{4.50}$$

We choose $p(\mathbf{x}_a \mid \mathbf{x})$ to be a uniform distribution over rotated images, leading to a rotational invariance, and use the variational lower bound to train the bounds of rotation α. To perform well on this task, we expect the recovered bounds α to be at least as large as the true value α_{true} to account for the rotational invariance. Too large values, i.e. $\alpha \approx 180°$, should be avoided due to ambiguities between, for example, 6s and 9s.

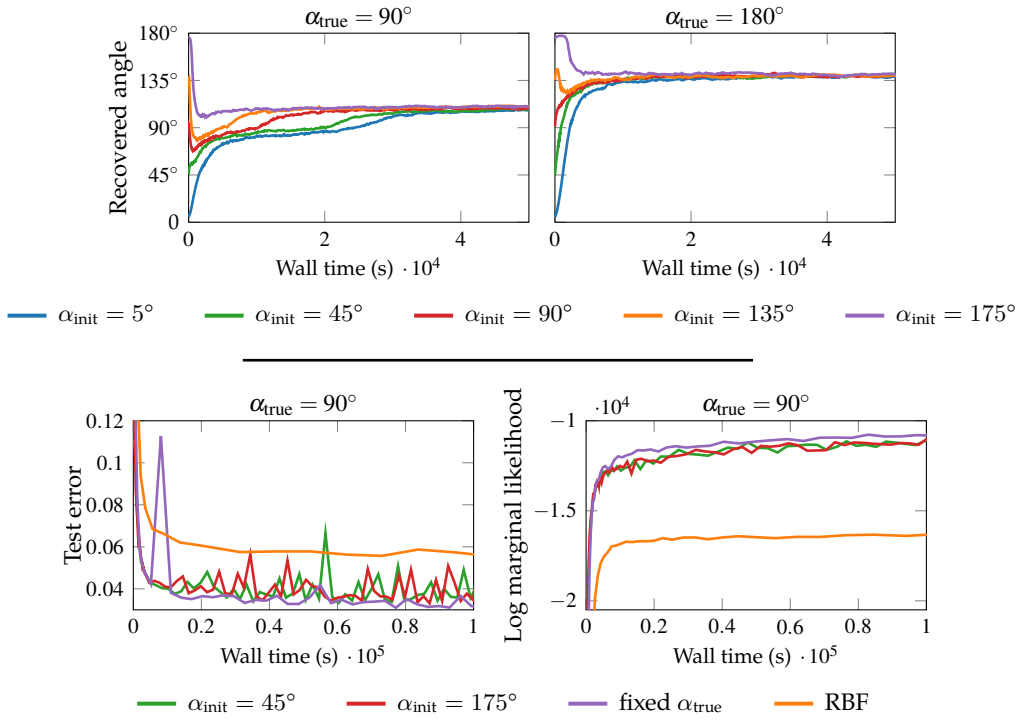


Figure 4.2: Binary classification on the partially rotated (by up to $\pm 90°$ or $\pm 180°$) MNIST dataset. *Top:* Recovered angles during optimisation for different initialisations α_{init} for the two cases $\alpha_{\text{true}} = 90°$ and $\alpha_{\text{true}} = 180°$. *Bottom:* Evolution of the error on a test set and the log marginal likelihood bound throughout the optimisation.

We find that the trained GP models with invariances are able to approximately recover the true angles (Figure 4.2, *top*). When $\alpha_{\text{true}} = 180°$, the angle is under-estimated, whereas $\alpha_{\text{true}} = 90°$ is recovered well. Regardless, all models outperform the simple RBF GP by a large margin, both in terms of error, and in terms of the marginal likelihood bound (Figure 4.2, *bottom*). These results show that our approach can be combined effectively with certain non-Gaussian likelihood models using the Pòlya-Gamma trick. Note that the marginal likelihood also identifies the "correct" angle as it assigns the best (highest) values to the model with angle fixed to α_{true}.

4.5.3 *Classification of MNIST digits*

Next, we consider full MNIST classification using a Gaussian likelihood, and compare the non-invariant RBF kernel to various invariant kernels. Figure 4.3 shows that the GPs with invariant kernels clearly outperform the baseline RBF kernel. Both types of learned invariances, affine transformations and local deformations, lead to similar performance for a wide range of initial conditions. When constrained to rotational invariances only, the results are only slightly better than the baseline. This indicates that deformations (stretching, shear) are more important than rotations for MNIST. Crucially, we do not require a validation set, but can use the log marginal likelihood of the training data to monitor performance. Table 4.1 lists the final test errors for the different models. In Figure 4.4 we show samples from $p(\mathbf{x}_a \mid \mathbf{x})$ for the trained model that uses all affine transformations in its covariance function.

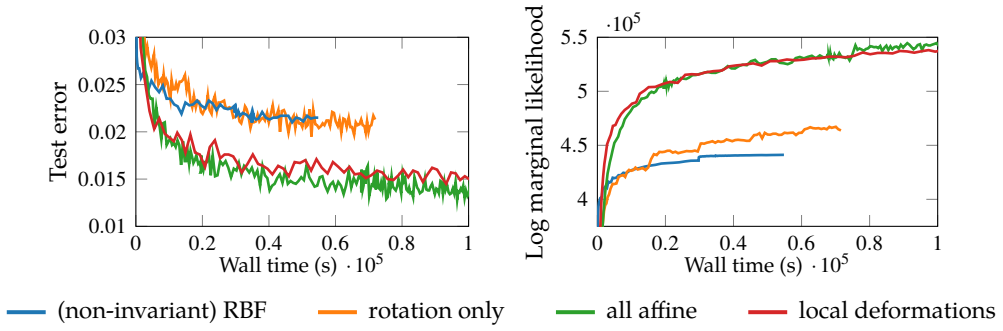


Figure 4.3: MNIST classification results using a Gaussian likelihood. *Left:* Test error. *Right:* Log marginal likelihood bound. All invariant models outperform the RBF baseline.

Invariances of kernel	**Error in %**
(non-invariant) RBF	2.15 ± 0.03
only rotations	2.08 ± 0.06
all affine transformations	1.35 ± 0.07
local deformations	1.47 ± 0.05

Table 4.1: Final test error for MNIST classification results using a Gaussian likelihood.

4.5.4 *Classification of rotated MNIST digits*

We also consider the fully rotated MNIST dataset[1]. In this case, we only run GP models that are invariant to affine transformations. In Figure 4.5 we compare

1 http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations

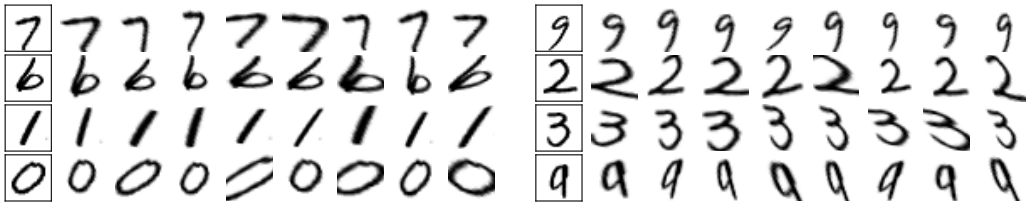


Figure 4.4: Samples from $p(\mathbf{x}_a \mid \mathbf{x})$ describing the *learned invariance* for eight example MNIST digits $\mathbf{x}$ (squares). The method becomes insensitive to the rotations, shears, and rotations that are present in the training set.

general affine transformations (learning bounds for all six affine parameters), rotations with learned angle bounds (no scalings, shears, or translations), and fixed rotational invariance. We found that all invariant models outperform the baseline (RBF) by a large margin. However, the models with fixed angles (no free parameters in the transformation) outperform their learned counterparts. We attribute this to the optimisation dynamics, as the problem of optimising the variational, kernel, and transformation parameters jointly is more difficult than optimising only variational and kernel parameters for fixed transformations. We emphasise that the marginal likelihood bound does correctly identify the best-performing invariance in this case as well (Figure 4.5 *(right)*).

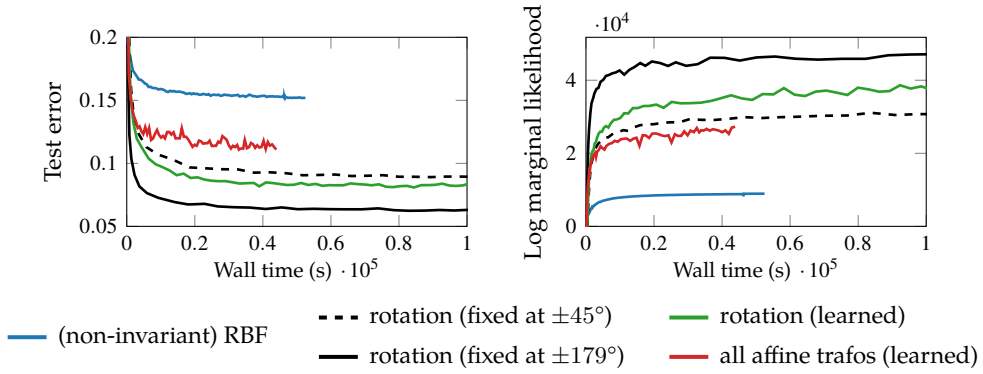


Figure 4.5: Rotated MNIST classification results. *Left:* Test error. *Right:* Log marginal likelihood bound. The optimiser has difficulty finding a good solution with the learned invariances, although the marginal likelihood bound correctly identifies the best model.

4.6 DISCUSSION AND OUTLOOK

As we have shown both theoretically and empirically, general invariances or insensitivities can indeed be encoded in the GP covariance function and provide a suitable inductive bias for learning. In all cases considered, a GP prior model with built-in invariances was superior to a GP prior without such invariances. Moreover, the bound to the log marginal likelihood correctly identified the "right"

invariances and could be used for model selection but also as an objective function to *learn* good parameter settings of invariances expressed through differentiably parameterisable transformations.

However, evaluating the bound to the log marginal likelihood entails several approximations and sub-sampling operations, which renders training time-consuming and limits it to a small number of jointly optimised transformation parameters due to optimisation difficulties.

Consequently, we restricted ourselves to simple invariances and insensitivities to define the augmentation sets: affine transformations and local deformations for which we learned global scales or ranges for the entire dataset. This choice constitutes a much stronger inductive biases, as we restrict the class of functions that can be modelled further and limit learnable invariances to this simple (yet effective) set.

In principle, we could choose much richer augmentation sets or, equivalently, augmentation distributions $p(\mathbf{x}_a \mid \mathbf{x})$. As a first step, we could make the transformations and deformations input-dependent instead of global; instead of learning global parameters that define the augmentations, we could amortise them using a recognition network and make them datapoint-dependent, similar to a VAE where the encoder outputs the parameters of a diagonal Gaussian posterior for each datapoint. Spatial transformer networks (Jaderberg et al., 2015) also employ a similar approach to infer datapoint-dependent affine parameters. As long as the amortised distribution is reparameterisable, we can backpropagate into these parameters.

More generally, we could move away from predefined interpretable transformations, such as rotations, and use conditional deep generative models directly to parameterise $p(\mathbf{x}_a \mid \mathbf{x})$. In other words, we could use a deep model that takes an image $\mathbf{x}$ as input, and outputs transformed or augmented images $\mathbf{x}_a$. Importantly, the model needs to output an entire distribution of images – or samples thereof – and not just a single point, which rules out the simplest image-to-image models (Ronneberger et al., 2015). Interesting candidates could be image to image model with conditional generative adversarial networks (GANs) (Isola et al., 2017), which allow us to inject randomness into the generative process.

As mentioned above, the principle obstacles to this approach of parameterising $p(\mathbf{x}_a \mid \mathbf{x})$ through such a flexible model are the large number of trainable parameters leading to unstable and difficult training. Moreover, deep generative models also entail longer sampling times – and therefore longer training times – compared to affine transformations or local deformations.

Here, we resorted to affine transformations and local deformations because of the entailed inductive biases. These allow us to express suitable invariances and insensitivities for the considered datasets and tasks through a modest number of

learnable parameters; especially in image classification tasks, it makes intuitive sense that small local deformations, rotations and scalings should not affect the label. Nevertheless, we learn the amount of transformations we should be invariant to. This is in stark contrast to other data augmentation techniques that rely on crossvalidation and expert knowledge (Loosli et al., 2007). Our results therefore highlight the role inductive biases play in learning.

4.7 SUMMARY

In this chapter we showed how invariances described by general data transformations can be incorporated into Gaussian process models.

We used "double-sum" kernels, which sum a base kernel over all points that are similar under the invariance. These kernels cannot be evaluated in closed form, due to integrals or a prohibitively large number of terms in the sums. Our method solves this technical issue by constructing a variational lower bound which only requires unbiased estimates of the kernel. Crucially, this variational lower bound also allowed us to learn good invariances by maximising the marginal likelihood bound through backpropagation through the sampling procedure.

We showed experimentally that our method can learn useful invariances for variants of MNIST. In some experiments, the performance of the joint optimisation problem was poorer than cases where the method was initialised with the correct invariance. Despite this drawback, the objective function correctly identified the best solution.

While we focused on kernels with invariances, we hope that our demonstration of learning with kernels that do not admit a closed-form evaluation will prove to be more generally useful by increasing the set of kernels with interesting generalisation properties that can be used.

Part II

PROBABILISTIC INFERENCE IN VARIATIONAL AUTOENCODERS

5

RESAMPLED PRIORS FOR VARIATIONAL AUTOENCODERS

So far, we have discussed supervised learning and probabilistic inference in Gaussian process models as well as the roles of inductive biases and priors in Gaussian process regression and classification. We now move on to study the role of the prior in variational autoencoders for unsupervised learning.

As we introduced in Section 1.6, variational autoencoders (VAEs) (P. and Welling, 2014; Rezende et al., 2014) are an example of continuous latent variable models. They are powerful and widely used probabilistic deep generative models, which map an often simple prior distribution on a low-dimensional latent space to complex distribution in a high-dimensional output space, to match a rich data distribution. They are trained using an encoder that amortises inference for the approximate posterior distribution in the latent space.

In their original formulation, both the prior and the variational posterior are parameterised by factorised Gaussian distributions. Many approaches have proposed using more expressive distributions to overcome these limiting modelling choices; most of these focus on more flexible variational approximate posterior distributions, see the discussion in Section 1.6.2. More recently, both the role of the prior and its mismatch to the *aggregate* variational posterior have been investigated in detail (Hoffman and Johnson, 2016; Rosca et al., 2018).

Here, we present *Learned Accept/Reject Sampling* (Lars), an alternative way to address the mismatch between the prior and the aggregate posterior in VAEs. The proposed Lars prior is the result of applying a rejection sampler with a learned acceptance function to the original prior, which now serves as the proposal distribution.

The results in this chapter are joint work with Andriy Mnih and were first published as 'Resampled Priors for Variational Autoencoders' (Bauer and Mnih, 2019). My main contributions were to jointly develop the idea, and individually perform the mathematical analysis and derivations as well as design and perform all experiments.

5.1 PROBLEM STATEMENT

In Section 1.6 we introduced VAEs as continuous latent variable models that are trained using the evidence lower bound to the (log) marginal likelihood. There, we also discussed the role of their priors and that the ELBO over the entire training set can be rewritten in terms of the empirical aggregate posterior $q(\mathbf{z}) = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})} q(\mathbf{z} \mid \mathbf{x})$ (Hoffman and Johnson, 2016), see Equation (1.48).

Recently, Tomczak and Welling (2018) noted that this formulation implies that the best prior $p(\mathbf{z})$ would correspond to the aggregate posterior – this would minimise the KL term in Equation (1.48) – though this would also overfit to the training set. Rosca et al. (2018) discussed the KL gap between aggregate posterior and prior and report a mismatch between the two for many generative probabilistic models.

We now discuss these arguments in more detail and then present our approach to address this mismatch.

In Figure 5.1 we illustrate the mismatch for a standard VAE with Gaussian approximate posterior and prior and a two-dimensional latent space on the MNIST dataset. Using the trained encoder model, we embed the empirical training distribution

$$\widehat{p}_{\text{Data}}(\mathbf{x}) = \frac{1}{N} \sum_n \delta(\mathbf{x} - \mathbf{x}_n) \tag{5.1}$$

into the latent space to obtain the empirical aggregate approximate posterior

$$\widehat{q}(\mathbf{z}) = \frac{1}{N} \sum_n q(\mathbf{z} \mid \mathbf{x}_n). \tag{5.2}$$

Here, we made the distinction between the distributions $p_{\text{Data}}(\mathbf{x})$ and $q(\mathbf{z})$ and their empirical counterparts evaluated on the training set, $\widehat{p}_{\text{Data}}(\mathbf{x})$ and $\widehat{q}(\mathbf{z})$, respectively. In the following, we usually mean the empirical distributions when referring to the data distribution or the aggregate posterior.

In our example, the aggregate posterior is given by a Gaussian mixture with $50,000$ components. We contrast it with the Gaussian prior and note the mismatch between the distributions: There are regions of the latent space to which the prior assigns relatively high probabilities whereas the aggregate posterior does not cover these regions at all.

Alternatively, we can evaluate the KL divergence between the aggregate posterior and the prior, $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z}))$, through MC sampling. Note that this example is only illustrative in that this model does not achieve state of the art performance. Rosca et al. (2018) perform a comprehensive analysis for more competitive models

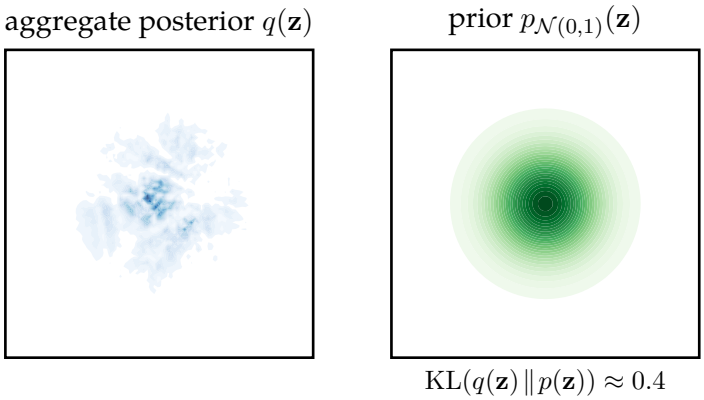


Figure 5.1: Illustrative example of the mismatch between aggregate posterior and prior. We show the aggregate posterior and fixed standard Normal prior for a regular VAE trained with the standard Normal prior on dynamic MNIST for a VAE with two-dimensional latent space. Darker colours correspond to higher densities.

with higher dimensional latent spaces with the same qualitative result: There exists a mismatch between the aggregate posterior and the prior for VAE type models.

This phenomenon is also sometimes referred to as "holes in the aggregate posterior", referring to regions of the latent space that have high density under the prior but very low density ("holes") under the aggregate posterior. These regions are almost never encountered during training, as the decoder or likelihood part of the VAE is only trained on samples from the approximate posterior. Reconstructed samples from these low density regions are typically decoded to observations that do not lie on the data manifold (Rosca et al., 2018). We show sample means from the matched and un-matched regions of the latent space for a simple VAE trained on dynamic MNIST in Figure 5.2.

Figure 5.2: Decoded sample means from the matched region *(left)* versus sample means from the mismatched region *(right)*. Note that these images come from a VAE with higher-dimensional latent space than in the illustrative Figure 5.1.

It has been shown that the prior minimising the KL term in the ELBO is given by the corresponding aggregate posterior (Tomczak and Welling, 2018); however, this choice usually leads to overfitting as this kind of non-parametric prior essentially memorises the training set and might not generalise beyond that. As we discussed in Section 1.5 the prior in continuous latent variable models predominantly acts as a (simple) base distribution that is embedded into a high-dimensional target

or output space by the decoder model. In effect, it regularises the latent space and can lead to underfitting or over-regularised models. Using more flexible and learnable priors corresponds to changing the probabilistic model; this may not make the model more interpretable but empirically improves performance.

The VampPrior (Tomczak and Welling, 2018) models the aggregate posterior explicitly using a mixture of posteriors of learned virtual observations $\{\mathbf{u}_m\}_m$ with a fixed number of components M,

$$p_{\text{Vamp}}(\mathbf{z}) = \sum_{m=1}^{M} q(\mathbf{z} \mid \mathbf{u}_m). \tag{5.3}$$

The virtual observations $\mathbf{u}_m$ are jointly optimised during training similar to how inducing inputs are optimised in sparse GP approximations. Tomczak and Welling (2018) find that for simple greyscale images the learned virtual observations can represent noisy prototypes of classes, such as MNIST digits.

Other ways of specifying expressive priors include the Gaussian process density sampler (Murray et al., 2009), as well as flow-based (Chen et al., 2017) and autoregressive models (Gulrajani et al., 2017), which provide different trade-offs between expressiveness and efficiency.

In the remainder of this chapter we introduce *Learned Accept/Reject Sampling* (Lars) to specify flexible prior distributions. In Section 5.10 we compare our approach to other approaches in the literature including those mentioned above.

5.2 OVERVIEW OF THE PROPOSED SOLUTION

Our proposed Lars prior is the result of applying a rejection sampler with a learned acceptance function to the original prior, which now serves as the proposal distribution.

The acceptance function, typically parameterised using a neural network, can be learned either jointly with the rest of the model by maximising the ELBO, or separately by maximising the likelihood of samples from the aggregate posterior of a trained model. Our approach is orthogonal to most current approaches for making priors more expressive, such as flow based priors, and can be easily combined with them by using them as proposals for the sampler. The price we pay for richer priors is iterative sampling, which can require rejecting a large number of proposals before accepting one. To improve the acceptance rate and, thus, the sampling efficiency of the sampler, we introduce a truncated version that always accepts the next proposal if some predetermined number of samples have been rejected in a row.

There is nothing inherent to stop Lars priors from overfitting, similar to other flexible and learnable priors. Our intuition is that by not making the parameterised

acceptance function too flexible, it will focus on rejecting larger regions of the latent space rather than modelling random noise, see also $a(\mathbf{z})$ in Figure 5.3. Ultimately, the question of overfitting and the flexibility of the prior depends on the problem at hand. We further discuss it in our ablation studies in Section 5.7.

5.3 LEARNED ACCEPT/REJECT SAMPLING

Learned accept/reject sampling (LARS) is a practical method to approximate a target density $q(\mathbf{z})$ from which we can sample but whose density is either too expensive to evaluate or unknown. It works by reweighting a simpler proposal distribution $\pi(\mathbf{z})$ using a learned acceptance function $a(\mathbf{z})$ as visualised in Figure 5.3. The aggregate posterior of a VAE, which is a very large mixture, is one example of such a target distribution. For simplicity, we focus on continuous variables in this section; however, our approach is equally applicable to discrete random variables as we briefly explain in Section 5.9.

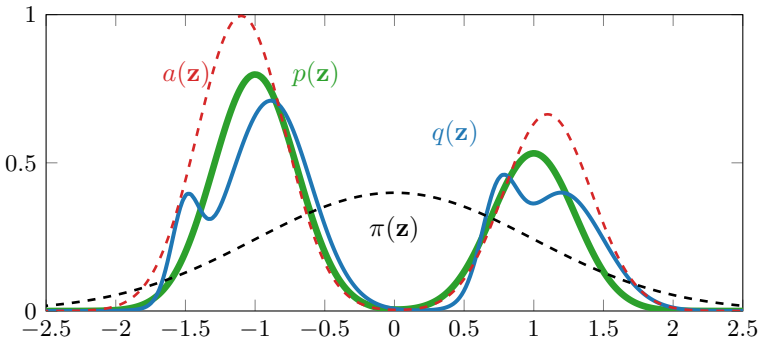


Figure 5.3: LARS approximates a target density $q(\mathbf{z})$ (——) by a resampled density $p(\mathbf{z})$ (——) obtained by multiplying a proposal $\pi(\mathbf{z})$ (- - -) with a learned acceptance function $a(\mathbf{z}) \in [0, 1]$ (- - -) and normalising.

LARS can be related to energy based models (Lecun et al., 2006) by viewing the acceptance function as the exponential of an energy. In a follow up work to the research presented in this chapter, Lawson et al. (2019) explore this connection to energy based models further.

In a similar vein to LARS, the Gaussian Process density sampler (Murray et al., 2009) also models a density by transforming a prior or proposal density. It uses a squashed latent GP function to reweight the proposal and allows for unbiased sampling through rejection sampling.

We further discuss these and other connections to related work in Section 5.10

5.3.1 *Indefinite rejection sampling*

Our goal is to approximate a complex target distribution $q(\mathbf{z})$ that we can sample from but whose density we are unable to evaluate. We start with an easy-to-sample-from base distribution $\pi(\mathbf{z})$ with a tractable density with the same support as $q(\mathbf{z})$, which might have learnable parameters but is insufficiently expressive to approximate $q(\mathbf{z})$ well. We then transform $\pi(\mathbf{z})$ into a more expressive distribution $p_\infty(\mathbf{z})$ by multiplying it by an *acceptance function* $a_\lambda(\mathbf{z}) \in [0, 1]$ and renormalising the product to obtain the density

$$p_\infty(\mathbf{z}) = \frac{\pi(\mathbf{z})a_\lambda(\mathbf{z})}{Z}; \qquad Z = \int \pi(\mathbf{z})a_\lambda(\mathbf{z})\,\mathrm{d}\mathbf{z}, \tag{5.4}$$

where Z is the normalisation constant and λ are the learnable parameters of $a_\lambda(\mathbf{z})$; we will drop the subscript λ in the following.

Proposition 5.1 (Indefinite rejection sampling)
$p_\infty(\mathbf{z})$ is also the distribution we obtain by using rejection sampling (von Neumann, 1951) with $\pi(\mathbf{z})$ as the proposal and $a(\mathbf{z})$ as the acceptance probability function.

Proof.
By definition, the probability of sampling $\mathbf{z}$ from the proposal is given by $\pi(\mathbf{z})$, so the probability of sampling and then accepting a particular $\mathbf{z}$ is given by the product $\pi(\mathbf{z})a(\mathbf{z})$. The probability of sampling and then rejecting $\mathbf{z}$ is given by the complementary probability $\pi(\mathbf{z})(1 - a(\mathbf{z}))$.

Thus, the probability of accepting any sample is given by $\int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z}$, whereas the probability of rejecting any sample is given by the complement $1 - \int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z}$.

Because samples are i.i.d., the probability of accepted samples $p_\infty(\mathbf{z})$ is given by:

$$p_\infty(\mathbf{z}) = \Pr(\mathbf{z} \mid \text{accepted}) = \frac{\Pr(\text{accepted}, \mathbf{z})}{\Pr(\text{accepted})} \tag{5.5}$$

$$= \frac{\pi(\mathbf{z})a(\mathbf{z})}{\int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z}} \tag{5.6}$$

Consequently, the log probability is given by:

$$\log p(\mathbf{z}) = \log \pi(\mathbf{z}) + \log a(\mathbf{z}) - \log Z \tag{5.7}$$

Because $0 \leq a(\mathbf{z}) \leq 1$ and $\pi(\mathbf{z})$ is normalised, we find that $0 \leq Z \leq 1$. □

Proof (Alternative proof from sampling).
Alternatively, the density $p_\infty(\mathbf{z})$ for sampling and accepting a particular sample $\mathbf{z}$ can also be derived as the sum over an infinite series of events: We

could sample $\mathbf{z}$ in the first draw and accept it; or we could reject the sample in the first draw and subsequently sample and accept $\mathbf{z}$; and so on. As all draws are independent, we can compute the sum as a geometric series:

$$p_\infty(\mathbf{z}) = \sum_{t=1}^{\infty} \Pr\left(\text{accept } \mathbf{z} \text{ at step } t \,|\, \text{rejected previous } t-1 \text{ samples}\right) \times \Pr\left(\text{reject } t-1 \text{ samples}\right) \tag{5.8}$$

$$= \sum_{t=1}^{\infty} a(\mathbf{z})\pi(\mathbf{z}) \left(1 - \int \pi(\mathbf{z})a(\mathbf{z})\right)^{t-1} \mathrm{d}\mathbf{z} \tag{5.9}$$

$$= \pi(\mathbf{z})a(\mathbf{z}) \sum_{t=0}^{\infty} (1-Z)^t; \qquad Z = \int \pi(\mathbf{z})a(\mathbf{z})\ \mathrm{d}\mathbf{z} \tag{5.10}$$

$$= \pi(\mathbf{z})a(\mathbf{z}) \frac{1}{1-(1-Z)} \tag{5.11}$$

$$= \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}. \tag{5.12}$$

Note that going from Equation (5.9) to Equation (5.10) we have redefined the index of summation to obtain the standard formula for the sum of a geometric series:

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \qquad \text{for } |x| < 1 \tag{5.13}$$

□

Using Proposition 5.1, we can sample from p_∞ as follows: draw a candidate sample $\mathbf{z}$ from the proposal π and accept it with probability $a(\mathbf{z})$; if accepted, return $\mathbf{z}$ as the sample; otherwise repeat the process. This means we can think of $a(\mathbf{z})$ as a filtering function which – by rejecting different fractions of samples at different locations – can transform $\pi(\mathbf{z})$ into a distribution closer to $q(\mathbf{z})$. In fact, as we show in Section 5.3.5, if the capacity of $a(\mathbf{z})$ is unlimited, we can obtain a resampled distribution $p_\infty(\mathbf{z})$ that matches the target distribution perfectly.

The efficiency of the sampler is closely related to the normalising constant Z as stated in Proposition 5.2. Therefore, Z can be interpreted as the average probability of accepting a candidate sample.

Proposition 5.2 (Efficiency of indefinite rejection sampling)
On average, indefinite rejection sampling has an efficiency of Z. That is, we will accept every $1/Z$th sample on average.

Proof.
We start by noting that the number of resampling steps performed until a sample is accepted follows the geometric distribution with success probability Z:

$$\Pr(\text{resampling time} = t) = Z(1 - Z)^{t-1} \tag{5.14}$$

This is due to the fact that each proposed candidate sample is accepted with probability Z, such decisions are independent, and the process continues until a sample is accepted. As the expected value of a geometric random variable is simply the reciprocal of the success probability, the expected number of resampling steps is $\langle t \rangle_{p_\infty} = 1/Z$. □

We note that scaling $a(\mathbf{z})$ by a constant, while keeping it in the $[0, 1]$ range, has no effect on the resulting density $p_\infty(\mathbf{z})$ because Z gets scaled accordingly. Therefore, the same distribution can be induced by rejection samplers with very different efficiencies. This means that learning an efficient sampler might require explicitly encouraging $a(\mathbf{z})$ to be as large as possible.

The results in Propositions 5.1 and 5.2 are well known, e.g. (von Neumann, 1951; Murray et al., 2009), and we provide their derivations for completeness.

5.3.2 *Truncated rejection sampling*

We now describe a simple and effective way to guarantee a minimum sampling efficiency that uses a modified density p_T instead of p_∞. We derive it from the following *truncated sampling scheme*: instead of allowing an arbitrary number of candidate samples to be rejected before accepting one, we cap this number and always accept the T^{th} sample if the preceding $T - 1$ samples have been rejected. To our knowledge, these results are novel at least within the wider machine learning community.

Proposition 5.3 (Truncated rejection sampling)
The corresponding density of this truncated sampling scheme is a mixture of $p_\infty(\mathbf{z})$ and the proposal $\pi(\mathbf{z})$, with the mixing weights determined by the average rejection probability $(1 - Z)$ and the truncation parameter T:

$$p_T(\mathbf{z}) = (1 - \alpha_T)\frac{a(\mathbf{z})\pi(\mathbf{z})}{Z} + \alpha_T \pi(\mathbf{z}), \tag{5.15}$$

where $\alpha_T = (1 - Z)^{T-1}$.

Proof.
As for the proof of Proposition 5.1, we can derive the density in closed form, this time by utilising the formula for the truncated geometric series,

$$\sum_{n=0}^{N} x^n = \frac{1 - x^{N+1}}{1 - x} \qquad \text{for } |x| < 1 \tag{5.16}$$

$$\begin{aligned}
p_T(\mathbf{z}) &= \sum_{t=1}^{T} \Pr\left(\text{accept } \mathbf{z} \text{ at step } t \,|\, \text{rejected previous } t-1 \text{ samples}\right) \\
&\qquad \times \Pr\left(\text{reject } t-1 \text{ samples}\right) \\
&= \sum_{t=1}^{T-1} \Pr\left(\text{accept } \mathbf{z} \text{ at step } t \,|\, \text{rejected previous } t-1 \text{ samples}\right) \\
&\qquad \times \Pr\left(\text{reject } t-1 \text{ samples}\right) \\
&\qquad + \Pr\left(\text{accept } \mathbf{z} \text{ at step } T \,|\, \text{rejected previous } T-1 \text{ samples}\right) \\
&\qquad \times \Pr\left(\text{reject } T-1 \text{ samples}\right) \\
&= a(\mathbf{z})\pi(\mathbf{z}) \sum_{t=0}^{T-2} (1-Z)^t + \pi(\mathbf{z})(1-Z)^{T-1} \qquad (5.17) \\
&= a(\mathbf{z})\pi(\mathbf{z}) \frac{1-(1-Z)^{T-1}}{Z} + \pi(\mathbf{z})(1-Z)^{T-1}. \qquad (5.18)
\end{aligned}$$

Again, note that we have shifted the index of summation going from the second to third equality. Moreover, note that $a(\mathbf{z})$ does not occur in the second term in Equation (5.17) as we accept the T^{th} sample with probability 1 regardless of the actual value of $a(\mathbf{z})$. □

The obtained probability density is normalised, $\int p_T(\mathbf{z})\, \mathrm{d}\mathbf{z} = 1$, because $\int \pi(\mathbf{z})a(\mathbf{z})\, \mathrm{d}\mathbf{z} = Z$ and $\int \pi(\mathbf{z})\, \mathrm{d}\mathbf{z} = 1$. Therefore, the log probability is given by:

$$\log p_T(\mathbf{z}) = \log \pi(\mathbf{z}) + \log \left[a(\mathbf{z}) \frac{1-(1-Z)^{T-1}}{Z} + (1-Z)^{T-1} \right] \tag{5.19}$$

As mentioned above, p_T is a mixture of the untruncated density p_∞ and the proposal density π. We can consider the two limiting cases of $T = 1$ (accept every sample) and $T \to \infty$ (resample indefinitely) and recover the expected results:

$$\log p_{T=1}(\mathbf{z}) = \log \pi(\mathbf{z}) \tag{5.20}$$

$$\lim_{T\to\infty} \log p_T(\mathbf{z}) = \log \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}. \tag{5.21}$$

If we accept the first candidate sample ($T = 1$), we obtain the proposal, whereas if we sample *ad infinitum*, we converge to the result from Equation (5.7).

While the truncated sampling density will in general be less expressive due to smoothing by interpolation with π, its acceptance rate is guaranteed to be at least

$1/T$, with the expected number of resampling steps given by $(1-(1-Z)^T)/Z \leq T$; see Proposition 5.4. Thus, we can trade off approximation accuracy against sampling efficiency by varying T.

Proposition 5.4 (Efficiency of indefinite rejection sampling)
The expected number of resampling steps for the truncated resampling scheme is $(1-(1-Z)^T)/Z$

Proof.
We can derive the average number of resampling steps for truncated resampling using the result for indefinite resampling, $\langle t \rangle_{p_\infty} = 1/Z$, and noting that the geometric distribution is memoryless:

$$\langle t \rangle_{p_T} = \langle t \rangle_{p_\infty} - \langle t \rangle_{p_{T..\infty}} + \langle t \rangle_{\text{accept } T\text{th sample}} \tag{5.22}$$

$$= \frac{1}{Z} - \left(T - 1 + \frac{1}{Z}\right)(1-Z)^{T-1} + T(1-Z)^{T-1} \tag{5.23}$$

$$= \frac{1-(1-Z)^T}{Z} \tag{5.24}$$

where we used the memoryless property of the geometric series for the second term. Also note that the third term in Equation (5.23) does not include a factor of Z because the probability of accepting the T^{th} sample given that we rejected all previous $T-1$ samples is 1 instead of Z. □

From Proposition 5.4 we can derive the following limits and special cases, which agree with the intuitive behaviour for truncated resampling; specifically, if we reject all samples ($a(\mathbf{z}) \approx 0$, thus $Z \to 0^+$), the average number of resampling steps achieves the maximum possible value T, whereas if we accept every sample, it goes down to 1:

$$\langle t \rangle_{p_{T=1}} = 1 \tag{5.25}$$

$$\langle t \rangle_{p_{T=2}} = 2 - Z \leq 2 \tag{5.26}$$

$$\langle t \rangle_{p_{T\to\infty}} = \frac{1}{Z} \tag{5.27}$$

$$\lim_{Z\to 0^+} \langle t \rangle_{p_T} = T \tag{5.28}$$

$$\lim_{Z\to 1^-} \langle t \rangle_{p_T} = 1 \tag{5.29}$$

Using the truncated sampling density also has the benefit of providing a natural scale for $a(\mathbf{z})$ in an interpretable and principled manner as, unlike $p_\infty(\mathbf{z})$, $p_T(\mathbf{z})$ is not invariant under scaling $a(\mathbf{z})$ by $c \in (0, 1]$.

5.3.3 *Estimation of the normalisation constant Z*

Estimating the normalising constant $Z = \int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z}$ is the main technical challenge of our method. An estimate of Z is needed both for updating the parameters of $p_T(\mathbf{z})$ and for evaluating trained models. Note that during training the value of Z changes as $a(\mathbf{z})$ is adapted, whereas after training it is a scalar constant. Here, we give details about how to perform this estimation; the estimation process varies depending on whether we evaluate a model or train it.

FOR MODEL EVALUATION. For evaluation of the trained model, we estimate Z with a large number of Monte Carlo (MC) samples from the proposal:

$$Z_S = \frac{1}{S}\sum_{s}^{S} a(\mathbf{z}_s) \qquad \text{with } \mathbf{z}_s \sim \pi(\mathbf{z}). \tag{5.30}$$

In practice, we use $S = 10^{10}$ samples, and evaluating Z takes on the order of a few minutes on a GPU. The fast evaluation time is due to the relatively low dimensionality of the latent space, so drawing the samples and evaluating their acceptance function values $a(\mathbf{z})$ is fast and can be parallelised using large batch sizes (10^5). For symmetric proposals, we also use antithetic sampling, that is, if we draw $\mathbf{z}$, then we also include $-\mathbf{z}$, which also corresponds to a valid/exact sample from the proposal. Antithetic sampling can reduce variance (see, for example, Kroese et al. (2013, Section 9.3)).

In Figure 5.4 we show a representative example of how the Monte Carlo estimates of the normalisation Z typically evolve with the number of MC samples. We plot the estimate of Z as a function of the number of samples used to estimate it, S.

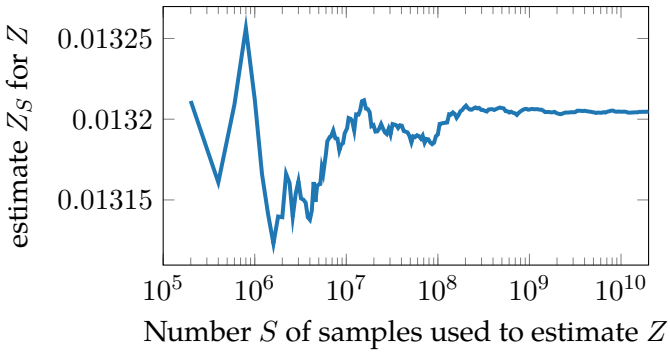


Figure 5.4: Estimation of Z by MC sampling from the proposal. For less than 10^7 samples the estimate is quite noisy but it stabilizes after about 10^9 samples.

FOR MODEL TRAINING. During training we would like to draw as few samples from the proposal as possible so as not to slow down training. In principle, we could use the same Monte Carlo estimate as introduced above in Equation (5.30). However, when using too few samples, the estimate for Z can be noisy, see Figure 5.4, and the values for $\log Z$ are substantially biased. In practice, we use two techniques to deal with these and related issues:

1. Smoothing the Z estimate by using exponentially moving averages in the forward pass, and

2. Including the sample from $q(\mathbf{z} \mid \mathbf{x})$, on which we evaluate the resampled prior to compute the KL in the objective function, in the estimator (similar to (Botev et al., 2017)).

We now describe these techniques in more detail.

During training we evaluate $\mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z}))$ in a stochastic fashion, that is, we evaluate $\log q(\mathbf{z}_r \mid \mathbf{x}_r) - \log p(\mathbf{z}_r)$ for each data point $\mathbf{x}_r$ in the minibatch and average the results. Here, $\mathbf{z}_r \sim q(\mathbf{z} \mid \mathbf{x}_r)$ is a sample from the variational posterior that corresponds to data point $\mathbf{x}_r$. Thus, we need to evaluate the resampled prior $p(\mathbf{z}_r)$ on all samples $\mathbf{z}_r$ in that batch. The mean KL contribution of the prior terms is then given by:

$$\frac{1}{R}\sum_{r=1}^{R}\log p_T(\mathbf{z}_r) = \frac{1}{R}\sum_{r}^{R}(\log \pi(\mathbf{z}_r) + \log a(\mathbf{z}_r)) - \log Z \tag{5.31}$$

where, so far, Z is shared between all data points in the minibatch. To reduce the variability of Z as well as the bias of $\log Z$, we smooth Z using exponentially moving averages. That is, given the previous moving average from iteration i, $\langle Z \rangle_i$, as well as the current estimate of Z computed from S MC samples, Z_i, we obtain the new moving average as:

$$\langle Z \rangle_{i+1} = (1-\epsilon)\langle Z \rangle_i + \epsilon Z_i \tag{5.32}$$

where ϵ controls how much Z is smoothed. In practice, we use $\epsilon = 0.1$ or $\epsilon = 0.01$. However, we can only use the moving average in the forward pass, as it is prohibitively expensive (and unnecessary) to backpropagate through all the terms in the sum. Therefore, we only want to backpropagate through the last term (Z_i), which needs to be rescaled to account for ϵ. Before we show how we do this, we introduce our second method for reducing variance: including the sample from $q(\mathbf{z} \mid \mathbf{x}_r)$ in the estimator.

For this, we introduce a datapoint dependent estimate Z_r, and replace the $\log Z$ term in Equation (5.31) by:

$$\log Z \to \frac{1}{R}\sum_r^R \log Z_r \tag{5.33}$$

$$Z_r = \frac{1}{S+1}\left[\sum_s^S a(\mathbf{z}_s) + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r \mid \mathbf{x}_r)} a(\mathbf{z}_r)\right] \qquad \mathbf{z}_s \sim \pi(\mathbf{z}); \mathbf{z}_r \sim q(\mathbf{z} \mid \mathbf{x}_r) \tag{5.34}$$

$$= \frac{1}{S+1}\left[S Z_S + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r \mid \mathbf{x}_r)} a(\mathbf{z}_r)\right] \tag{5.35}$$

where the sample $\mathbf{z}_r$ from the variational posterior is reweighted by the ratio of prior and variational posterior, similar to Botev et al. (2017). Note that we do not want to backpropagate through this fraction, as this would alter the gradients for the encoder and the proposal. We are now in a position to write down the algorithm that computes the different Z_r values to use in the objective, see Algorithm 1.

Algorithm 1: Estimation of $\log Z$ (Equation (5.33)) in the objective during training

input : Minibatch of samples $\{\mathbf{z}_r\}_r^R$ from $\{q(\mathbf{z} \mid \mathbf{x}_r)\}_r^R$; S samples $\{\mathbf{z}_s\}_s^S$ from π; previous moving average $\langle Z \rangle_i$

output: Updated moving average $\langle Z \rangle_{i+1}$; $\log Z$ estimate for current minibatch $\{\mathbf{x}_r\}_r^R$

1 $Z_S \leftarrow \frac{1}{S}\sum_s^S a(\mathbf{z}_s)$

2 **for** $r \leftarrow 1$ **to** R **do**

3 $\quad Z_{r,\text{curr}} \leftarrow \frac{1}{S+1}\left[S Z_S + \texttt{stop\_grad}\left(\frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r)}\right) a(\mathbf{z}_r)\right]$

4 $\quad Z_{r,\text{smooth}} \leftarrow (1-\epsilon)\langle Z \rangle_i + \epsilon Z_{r,\text{curr}}$

5 $\quad Z_r \leftarrow Z_{r,\text{curr}} + \texttt{stop\_grad}(Z_{r,\text{smooth}} - Z_{r,\text{curr}})$

6 **end**

7 $\langle Z \rangle_{i+1} \leftarrow \frac{1}{R}\sum_r^R \texttt{stop\_grad}(Z_{r,\text{smooth}})$

8 $\log Z \leftarrow \frac{1}{R}\sum_r^R \log Z_r$

Note that in the forward pass (Algorithm 1, line 5) evaluates to $Z_{r,\text{smooth}}$, whereas in the backward direction (when computing gradients) it evaluates to $Z_{r,\text{curr}}$. Thus, we use the smoothed version in the forward pass and the current estimate for the gradients.

While Algorithm 1 estimates $\log Z$ used in the density of the indefinite/untruncated resampling scheme, we can easily adapt it to compute the truncated density as well, as we compute the individual Z_r, which can be used instead.

5.3.4 *Illustrative examples in 2D*

Before applying LARS to VAEs, we investigate its ability to fit a mixture of Gaussians (MoG) target distribution in 2D, see Figures 5.5 and 5.6. We use a standard Normal distribution as the proposal and train an acceptance function parameterised by a small MLP, to reweight π to q by maximising the supervised objective

$$\mathbb{E}_{q(\mathbf{z})}\left[\log p_\infty(\mathbf{z})\right] = \mathbb{E}_{q(\mathbf{z})}\left[\log \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}\right], \tag{5.36}$$

which amounts to maximum likelihood learning.

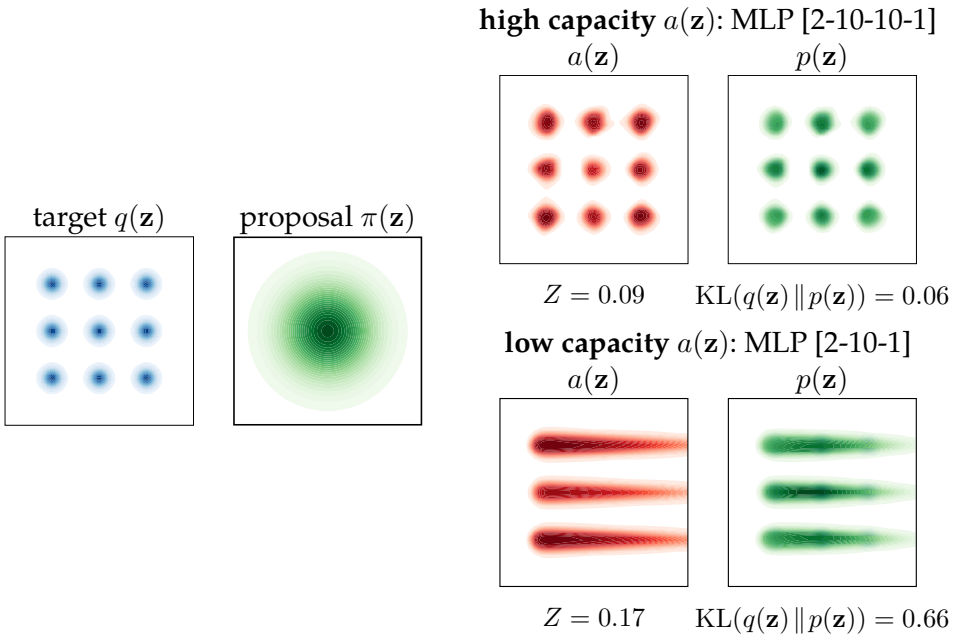


Figure 5.5: Illustrative 2D example: Learned acceptance functions $a(\mathbf{z})$ (red) that approximate a fixed target $q(\mathbf{z})$ (blue) by reweighting a proposal $\pi(\mathbf{z}) = \mathcal{N}(0, 1)$ to obtain an approximate density $p(\mathbf{z})$ (green). Darker colours correspond to higher densities.

We find that a small network for $a(\mathbf{z})$ (an MLP with two layers of 10 units each) is sufficient to separate out the individual modes almost perfectly (Figure 5.5 (*top*)). However, an even simpler network (one layer with 10 units) can already learn to cut away the unnecessary tails of the proposal and leads to a better fit than the proposal (Figure 5.5 (*bottom*)). While samples from the higher-capacity model are closer to (or have lower KL to) the target, the sampling efficiency is lower (smaller Z).

As a second example, we consider the same target distribution but use a RealNVP (Dinh et al., 2017) proposal which is trained jointly with the acceptance function, see Figure 5.6. Trained alone, the RealNVP warps the standard Normal distribution into a multi-modal distribution, but fails to isolate all the modes

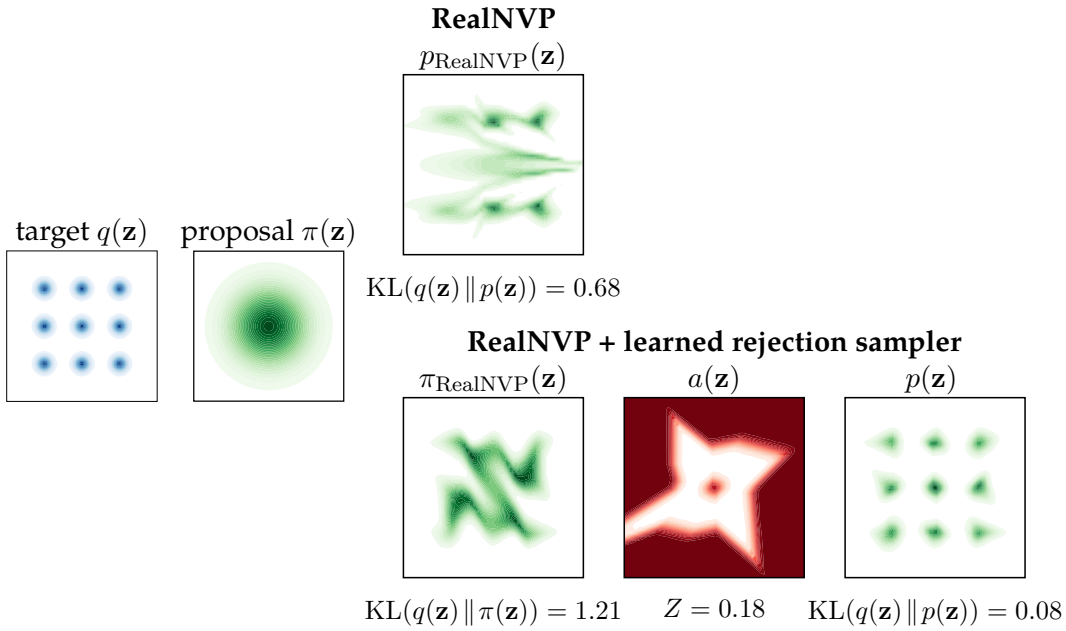


Figure 5.6: Illustrative 2D example when combining LARS with a RealNVP proposal. The target $q(\mathbf{z})$ is approximated by a proposal $\pi(\mathbf{z}) = \mathcal{N}(0, 1)$ that is either warped through a RealNVP alone, $p_{\text{RealNVP}}(\mathbf{z})$ (top), or warped by a RealNVP and then resampled with a learned rejection sampler (bottom). Darker colours correspond to higher densities.

(Figure 5.6 *(top)*, also shown by Huang et al. (2018)). On the other hand, the model obtained by using LARS with a RealNVP proposal manages to isolate all the modes. It also achieves a higher average acceptance probability than LARS with a standard Normal proposal, as the RealNVP proposal approximates the target better, see Figure 5.6 *(bottom)*.

5.3.5 *Relationship to classical rejection sampling*

Here we briefly review classical rejection sampling (von Neumann, 1951) and relate it to LARS; for an illustration of rejection sampling, see Figure 5.7. Rejection sampling provides a way to sample from a target distribution with a known density $q(\mathbf{z})$ which is hard to sample from directly but is easy to evaluate pointwise. Note that this is *exactly the opposite* of our setting, in which $q(\mathbf{z})$ is easy to sample from but infeasible to evaluate. By drawing samples from a tractable proposal distribution $\pi(\mathbf{z})$ and accepting them with probability $a(\mathbf{z}) = \frac{q(\mathbf{z})}{M\pi(\mathbf{z})}$ (where M is chosen so that $\pi(\mathbf{z})M \geq q(\mathbf{z}), \forall \mathbf{z}$), rejection sampling generates exact samples from $q(\mathbf{z})$. The average sampling efficiency, or acceptance rate, of this sampler is $1/M$, which means the less the proposal needs to be scaled to dominate the target, the fewer samples are rejected. In practice, it can be difficult to find the smallest valid M as $q(\mathbf{z})$ is usually a complicated multi-modal distribution.

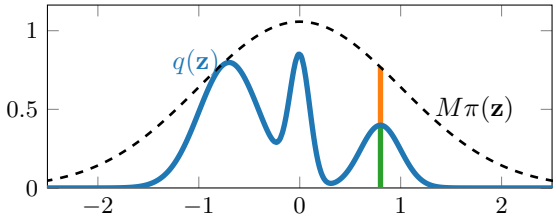


Figure 5.7: Classic rejection sampling enables sampling from a complicated target distribution. In classic rejection sampling we generate samples from a target distribution (——) by sampling from a simpler proposal (- - -) and accepting samples with probability $a(\mathbf{z}) = \frac{q(\mathbf{z})}{M\pi(\mathbf{z})}$. The green and orange line are proportional to the relative accept (——) and rejection (——) probability, respectively.

We can match $q(\mathbf{z})$ exactly with LARS by emulating classical RS and setting $a^*(\mathbf{z}) = c\frac{q(\mathbf{z})}{\pi(\mathbf{z})}$ with the constant c chosen so that $a^*(\mathbf{z}) \leq 1$ for all $\mathbf{z}$. This assumes that a^* has unlimited capacity. Noting that this results in $Z = c$ and comparing this to standard rejection sampling, we find that Z plays the role of $1/M$. However, with LARS our primary goal is learning a compact and fast-to-evaluate density that approximates $q(\mathbf{z})$, rather than constructing a way to sample from $q(\mathbf{z})$ exactly, which is already easy in our setting.

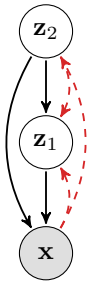
5.4 VAES WITH RESAMPLED PRIORS

While LARS can be used to enrich several distributions in a VAE, we concentrate on applying it to the prior; that is, we use the original VAE prior as a proposal distribution $\pi(\mathbf{z})$ and define a *resampled prior* $p(\mathbf{z})$ through a rejection sampler with learned acceptance probability $a(\mathbf{z})$. We parameterise $a(\mathbf{z})$ with a flexible neural network and use the terms "LARS prior" and "resampled prior" interchangeably.

In general, the acceptance function is trained jointly with the other parts of the model by optimising the ELBO objective (Equation (1.32)) but with the resampled prior instead of the standard Normal prior. This only modifies the prior term $\log p(\mathbf{z})$ in the objective, with $p(\mathbf{z})$ replaced by Equation (5.4) for indefinite resampling and by Equation (5.15) for truncated resampling. Thus, training a VAE with a resampled prior only requires evaluating the resampled prior density on samples from the variational posterior, as well as generating a modest number of samples from the *proposal* to update the moving average estimate of Z, as described above. Crucially, we never need to perform accept/reject sampling during training.

5.4.1 *Hierarchical models*

In addition to VAEs with a single stochastic layer and different encoder/decoder architectures, we also consider hierarchical VAEs with two stochastic layers. Similarly to Tomczak and Welling (2018), we apply the resampled prior to the top-most of these latent layers. We closely follow their proposed inference and generative structure as well as their architectural choices, such that the generative distribution $p_{\theta,\lambda}(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2)$ and the variational posterior $q_{\phi,\psi}(\mathbf{z}_1, \mathbf{z}_2 \mid \mathbf{x})$ factorise as stated in Figure 5.8.



$$p_{\theta,\lambda}(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2) = p_\theta(\mathbf{x} \mid \mathbf{z}_1, \mathbf{z}_2) p_\lambda(\mathbf{z}_1 \mid \mathbf{z}_2) p_\lambda(\mathbf{z}_2) \tag{5.37}$$

$$q_{\phi,\psi}(\mathbf{z}_1, \mathbf{z}_2 \mid \mathbf{x}) = q_\psi(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{z}_2) q_\phi(\mathbf{z}_2 \mid \mathbf{x}) \tag{5.38}$$

$$p_\lambda(\mathbf{z}_2) = p_T(\mathbf{z}_2) \tag{5.39}$$

$$p_\lambda(\mathbf{z}_1 \mid \mathbf{z}_2) = \mathcal{N}\big(\mathbf{z}_1 \mid \mu_\lambda(\mathbf{z}_2), \operatorname{diag} \sigma^2_\lambda(\mathbf{z}_2)\big) \tag{5.40}$$

$$q_\phi(\mathbf{z}_2 \mid \mathbf{x}) = \mathcal{N}\big(\mathbf{z}_2 \mid \mu_\phi(\mathbf{x}), \operatorname{diag} \sigma^2_\phi(\mathbf{x})\big) \tag{5.41}$$

$$q_\psi(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{z}_2) = \mathcal{N}\big(\mathbf{z}_1 \mid \mu_\psi(\mathbf{x}, \mathbf{z}_2), \operatorname{diag} \sigma^2_\psi(\mathbf{x}, \mathbf{z}_2)\big) \tag{5.42}$$

Figure 5.8: Hierarchical VAE with two latent spaces. Arrows indicate the generative model (⟶) and the inference model (- -➤), respectively. Graphical model reproduced from Tomczak and Welling (2018, Fig 1)

The means $\mu_{\{\mu,\phi,\psi\}}$ and log standard deviations $\log \sigma_{\{\mu,\phi,\psi\}}$ are parameterised by neural networks; we apply resampling to the top-most prior distribution $p_\lambda(\mathbf{z}_2)$.

5.4.2 *Training procedure for a VAE with resampled prior*

LARS constitutes another method to specify rich priors, which are parameterised through the acceptance function $a_\lambda(\mathbf{z})$; we use subscript λ in this section to highlight the learnable parameters of the acceptance function. The acceptance function only modifies the prior of the model with the encoder and decoder remaining unchanged. Thus, the only part of the training objective, which changes compared to training of a normal VAE, is the evaluation of the prior contribution to the KL term, $\log p(\mathbf{z})$. This log density is evaluated on samples from the encoder distribution $q_\varphi(\mathbf{z} \mid \mathbf{x})$. Both for truncated and untruncated sampling, evaluation of $\log p(\mathbf{z})$ entails evaluating

1. the proposal density $\pi(\mathbf{z})$
2. the acceptance function $a_\lambda(\mathbf{z})$
3. the normaliser Z

and their logarithms. Evaluation of the proposal and acceptance function is straightforward and evaluation of the normaliser and its logarithm are detailed in Algorithm 1.

To update the normaliser during training, we require a modest number of samples from the proposal; however, we never actually perform the accept/reject step and never need to decode these samples into image space using the decoder model.

The full training procedure for a VAE as well as the changes necessary due to the resampled prior are detailed in Algorithm 2. While we only describe the pseudo code for resampled priors with untruncated or indefinite resampling, an extension to the truncated case is straightforward. The changes compared to training a regular VAE with standard prior are limited to an evaluation of Algorithm 1 and a modification of the KL divergence computation in the objective.

Algorithm 2: Pseudo code for training of a VAE with resampled prior (for untruncated resampling)

input : dataset of images $\mathcal{D}_{\text{train}} = \{\mathbf{x}_n\}_n^N$
output: trained model parameters φ, θ (encoder/decoder) and $\lambda, \log Z$ (resampled prior)

initialize parameters of encoder φ and decoder θ
initialize parameters of the resampled prior λ and the moving average $\langle Z \rangle$
for $it \leftarrow 1$ **to** N_{it} **do**
  sample a minibatch of data $\{\mathbf{x}_r\}_r^R$
  sample latents $\{\mathbf{z}_r\}_r^R$ from the encoder distribution $\{q_\varphi(\mathbf{z} \mid \mathbf{x}_r)\}_r^R$ (using reparameterisation)
  evaluate the reconstruction term: recon $\leftarrow \frac{1}{R} \sum_r^R \log p_\theta(\mathbf{x} \mid \mathbf{z}_r)$
  update moving average $\langle Z \rangle$ and compute $\log Z$ using Algorithm 1
  evaluate the KL term:
  kl $\leftarrow \frac{1}{R} \sum_r^R [\log q_\varphi(\mathbf{z}_r \mid \mathbf{x}_r) - \log \pi(\mathbf{z}_r) - \log a_\lambda(\mathbf{z}_r)] + \log Z$
  evaluate the objective: recon $-$ kl
  backpropagate gradients into all parameters φ, θ and λ
end
compute final normaliser for evaluation: $Z \leftarrow \frac{1}{S_{\text{eval}}} \sum_s^{S_{\text{eval}}} a(\mathbf{z}_s)$ with $\mathbf{z}_s \sim \pi(\mathbf{z})$

5.5 EXPERIMENTAL SETUP

5.5.1 *Datasets*

We perform unsupervised learning on a suite of standard datasets: static and dynamically binarised MNIST (Larochelle and Murray, 2011), Omniglot (Lake et al., 2015), and FashionMNIST (Xiao et al., 2017).

The MNIST dataset (LeCun et al., 1998) contains 50,000 training, 10,000 validation and 10,000 test images of the digits 0 to 9. Each image has a size of 28×28 pixels. We use both a dynamically binarised version of this dataset as well as the statically binarised version introduced by Larochelle and Murray (2011). Dynamic binarisation is typically used to reduce the risk of overfitting.

Omniglot (Lake et al., 2015) contains 1,623 handwritten characters from 50 different alphabets, with differently drawn examples per character. The images are split into 24,345 training and 8,070 test images. Following Tomczak and Welling (2018) we take 1,345 training images as a validation set. Each image has a size of 28×28 pixels and we applied dynamic binarisation to them.

FashionMNIST (Xiao et al., 2017) is a recently proposed plug-in replacement for MNIST with 60,000 train/validation and 10,000 test images split across 10 classes. Each image has size of 28×28 pixels and we applied dynamic binarisation to them.

5.5.2 *Architectures/Models*

We consider several architectures:

1. a VAE with single latent layer and an MLP encoder/decoder (referred to as "VAE");
2. a VAE with two latent layers and an MLP encoder/decoder ("HVAE");
3. a VAE with two latent layers and a convolutional encoder/decoder ("ConvHVAE").

For the acceptance function we use an MLP network with two layers of 100 `tanh` units and the `logistic` non-linearity for the output. Moreover, we perform resampling with truncation after $T = 100$ attempts (Equation (5.15)), as a good trade-off between approximation accuracy and sampling efficiency. We explore alternative architectures and truncations in Section 5.7. Full details of all architectures can be found in Appendix A.

5.5.3 *Training and evaluation.*

All models were trained using the Adam optimiser (Kingma and Ba, 2015) for 10^7 iterations with the learning rate $3 \cdot 10^{-4}$, which was decayed to $1 \cdot 10^{-4}$ after 10^6 iterations, and mini-batches of size 128. Weights were initialised according to (Glorot and Bengio, 2010). Unless otherwise stated, we used warm-up (KL-annealing) for the first 10^5 iterations (Bowman et al., 2016). We quantitatively evaluate all methods using the negative test log likelihood (NLL) estimated using 1000 importance samples, see Section 1.6.3. We estimate Z using S MC samples (see Equation (5.30)) with $S = 10^{10}$ for evaluation after training (this only takes several minutes on a GPU) and $S = 2^{10}$ during training. We repeated experiments for different random seeds with very similar results, so report only the means. We observed overfitting on static MNIST, so on that dataset we performed early stopping using the NLL on the validation set and halved the times for learning rate decay and warm-up.

5.6 EMPIRICAL EVALUATION: QUANTITATIVE RESULTS

	VAE ($L = 1$)		**HVAE** ($L = 2$)		**ConvHVAE** ($L = 2$)	
DATASET	standard	LARS	standard	LARS	standard	LARS
staticMNIST	89.11	**86.53**	85.91	**84.42**	82.63	**81.70**
dynamicMNIST	84.97	**83.03**	82.66	**81.62**	81.21	**80.30**
Omniglot	104.47	**102.63**	101.38	**100.40**	97.76	**97.08**
FashionMNIST	228.68	**227.45**	227.40	**226.68**	226.39	**225.92**

Table 5.1: Test negative log likelihood (NLL; *lower is better*) for different models with standard Normal prior ("standard") and our proposed resampled prior ("LARS"). L is the number of stochastic layers in the model.

	standard $\mathcal{N}(0, 1)$				LARS *post-hoc*			
MODEL	ELBO	KL	recons	Z	ELBO	KL	recons	Z
dynamicMNIST	89.0	25.8	63.2	1	87.3	24.1	63.2	0.023
Omniglot	110.8	37.2	73.6	1	108.8	35.2	73.6	0.015
					LARS *joint training*			
					ELBO	KL	recons	Z
dynamicMNIST					86.6	24.7	61.9	0.015
Omniglot					108.4	35.9	72.5	0.011

Table 5.2: ELBO and its components (KL and reconstruction error) for VAEs with: *top:* a standard Normal prior, a *post-hoc* fitted LARS prior; *bottom:* a jointly trained LARS prior.

First, we compare the resampled prior to a standard Normal prior on several architectures, see Table 5.1. In all cases considered, ranging from a simple VAE to a hierarchical VAE with convolutional encoders/decoders, the resampled prior outperforms the standard prior. In most cases, the improvement in the NLL exceeds 1 nat and is notably larger for the simpler architectures. In other words, our experiments show that LARS priors consistently improve the performance of VAEs.

Applied *post-hoc* to a pretrained model (fixed encoder/decoder) the resampled prior almost reaches the performance of a jointly trained model, see Table 5.2. This highlights that LARS can also be effectively employed to improve trained models. In this case, the gain in ELBO is entirely due to a reduction in the KL as the reconstruction error is determined by the fixed encoder/decoder. In contrast, joint training reduces both, and reaches a better ELBO, though the reduction in KL is smaller than for post-hoc training.

MODEL	NLL
VAE ($L = 2$) + VGP (Tran et al., 2016)	81.32
LVAE ($L = 5$) (Sønderby et al., 2016)	81.74
HVAE ($L = 2$) + LARS prior	81.62
VAE + IAF (Kingma et al., 2016)	**79.10**
ConvHVAE + LARS prior	80.30

Table 5.3: Test NLL on dynamic MNIST for non-convolutional (above the line) and convolutional (below the line) models.

MODEL	NLL
IWAE ($L = 2$) (Burda et al., 2016)	103.38
LVAE ($L = 5$) (Sønderby et al., 2016)	102.11
HVAE[1] ($L = 2$) + VampPrior (Tomczak and Welling, 2018)	101.18
HVAE ($L = 2$) + LARS prior	**100.40**
ConvHVAE[1] ($L = 2$) + VampPrior (Tomczak and Welling, 2018)	97.56
ConvHVAE ($L = 2$) + LARS prior	97.08

[1] We use the same structure but a simpler architecture than Tomczak and Welling (2018).

Table 5.4: Test NLL on dynamic Omniglot.

In Tables 5.3 and 5.4 we compare LARS to other approaches on dynamic MNIST and Omniglot. LARS is competitive with other approaches that make the prior or variational posterior more expressive when applied to similar architectures. We expect that our results can be further improved by using PixelCNN-style decoders, which have been used to obtain state-of-the-art results on dynamic MNIST (78.45 nats for PixelVAE with VampPrior (Tomczak and Welling, 2018)) and Omniglot (89.83 with VLAE (Chen et al., 2017)). The improvements of LARS

over the standard Normal prior are generally comparable to those obtained by the VampPrior (Tomczak and Welling, 2018), though they tend to be slightly smaller. LARS priors and VampPriors have somewhat complementary strengths. Sampling from VampPriors is more efficient as it amounts to sampling from a mixture model, which is non-iterative. On the other hand, the VampPriors that achieve the above results use 500 or 1000 pseudo-inputs and thus have hundreds of thousands of parameters, whereas the number of parameters in the LARS priors is more than an order of magnitude lower. As LARS priors operate on a low-dimensional latent space they are also considerably more efficient to train than VampPriors, which are connected to the input space through the pseudo-inputs.

5.7 EMPIRICAL EVALUATION: ABLATION STUDIES

5.7.1 *Influence of network architecture*

We investigated different network sizes for the acceptance function. As for the illustrative example, even a simple architecture can already notably improve over a standard prior, with successively more expressive networks further improving performance, see Table 5.5. Notably, the average acceptance probability Z is influenced much more by the truncation parameter than the network architecture. In practice, we choose $a = \text{MLP}[100 - 100]$ as more expressive networks only showed a marginal improvement. While we suspect that substantially larger networks will lead to overfitting, we did not observe this for the networks considered in Table 5.5.

MODEL	NLL	Z
VAE ($L = 1$, MLP)	84.97	1
VAE + LARS; $a = \text{MLP}[10 - 10]$	84.08	0.02
VAE + LARS; $a = \text{MLP}[50 - 50]$	83.29	0.016
VAE + LARS; $a = \text{MLP}[100 - 100]$	83.05	0.015
VAE + LARS; $a = \text{MLP}[50 - 50 - 50]$	83.09	0.015
VAE + LARS; $a = \text{MLP}[100 - 100 - 100]$	82.94	0.014

Table 5.5: Test NLL and Z on dynamic MNIST. Different network architectures for $a(\mathbf{z})$ with $T = 100$.

5.7.2 *Influence of truncation*

The truncation parameter T tunes the trade-off between sampling efficiency and approximation quality. As shown in Table 5.6, the NLL improves as the number of steps before truncating increases, whereas the value for Z decreases accordingly.

The gains in the NLL come both from the KL as well as the reconstruction term, though the relative contribution from the KL is larger. We stress that for truncated sampling, Z denotes the average acceptance rate *for the first* $T-1$ *attempts* and does not take into account always accepting the T^{th} proposal. As a reference point, a model that is *trained* with indefinite (untruncated) resampling only accepts 2 out of 10^6 samples and estimating Z accurately thus requires a lot of samples.

MODEL	NLL	Z	KL	recons
VAE ($L=1$, MLP)	84.97	1	25.8	63.3
VAE + LARS; $p_{T=2}$	84.60	0.19	25.4	63.2
VAE + LARS; $p_{T=5}$	84.11	0.12	25.1	63.0
VAE + LARS; $p_{T=10}$	83.71	0.08	25.0	62.6
VAE + LARS; $p_{T=50}$	83.24	0.03	24.8	62.1
VAE + LARS; $p_{T=100}$	83.05	0.014	24.7	61.9
VAE + LARS; p_{∞}	82.67	$2 \cdot 10^{-6}$	24.8	61.5

Table 5.6: Influence of the truncation parameter on the test set of dynamic MNIST; $a = \text{MLP}[100-100]$.

5.7.3 *Combination with non-factorial priors*

So far, we have only considered VAEs with standard Normal priors. As stated before, our resampled prior is an orthogonal approach to specifying rich distributions and can be combined with other structured proposals such as non-factorial or autoregressive flow distributions. In the illustrative example in Section 5.3.4 we saw that the LARS prior can model discontinuous densities at the expense of potentially long sampling times, whereas flow-based methods are constrained to continuous densities but are very efficient. We combined the two to yield superior results at higher sampling efficiencies.

We investigated this using a RealNVP distribution as an alternative proposal on MNIST (Table 5.7) and Omniglot (Table 5.8). The RealNVP prior by itself outperforms our resampling approach applied to a standard Normal prior. However, applying LARS to a RealNVP proposal distribution, we obtain a further improvement of more than 0.5 nats on both MNIST and Omniglot. A VAE trained with a more expressive Masked Autoregressive Flow (MAF) (Papamakarios et al., 2017) as a prior outperformed the RealNVP prior but was still worse than a resampled RealNVP. Combining LARS with an MAF proposal is impractical due to slow sampling of the MAF.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	84.97	1
VAE + RealNVP prior	81.86	1
VAE + MAF prior	81.58	1
VAE + LARS $\mathcal{N}(0, 1)$	83.04	0.015
VAE + LARS RealNVP	**81.15**	0.027

Table 5.7: More expressive proposal and prior distributions on dynamic MNIST.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	104.53	1
VAE + RealNVP prior	101.50	1
VAE + resampled $\mathcal{N}(0, 1)$	102.68	0.012
VAE + resampled RealNVP	100.56	0.018

Table 5.8: More expressive proposal and prior distributions on dynamic Omniglot.

5.8 EMPIRICAL EVALUATION: QUALITATIVE RESULTS

5.8.1 *Samples from trained models*

For a qualitative analysis, we consider unconditional samples from a VAE with a resampled prior. Our learned acceptance function assigns probability values $a(\mathbf{z})$ to all points in the latent space. This allows us to effectively rank draws from the proposal and to compare particularly high- to particularly low-scoring ones.

We perform the same analysis twice, once for a resampled prior that is *trained jointly* with the VAE (Figure 5.9) and once for a resampled prior that is *trained post-hoc* for a fixed VAE (Figure 5.11). In both cases, we generated 10^4 samples from the proposal $\pi(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, 1)$ and sorted them by their acceptance value $a(\mathbf{z})$. Figures 5.9 and 5.11 show samples selected by their acceptance probability as assigned by the acceptance function $a(\mathbf{z})$. The top row shows samples with highest values (close to 1), which would almost certainly be accepted by the resampled prior; visually, these samples are very good. The middle row shows 25 random samples from the proposal, some of which have high visual quality while others show visible artifacts. The corresponding acceptance values typically reflect this, see also Figure 5.10, in which we also show the $a(\mathbf{z})$ distribution for all 10^4 samples from the proposal sorted by their a value. The bottom row shows samples with the lowest acceptance probability (typically below $a(\mathbf{z}) \approx 10^{-10}$), which would almost surely be rejected by the resampled prior. Visually, these samples are very poor. When accepting/rejecting samples according to our pre-defined resampling scheme, these low-quality samples would most likely be rejected.

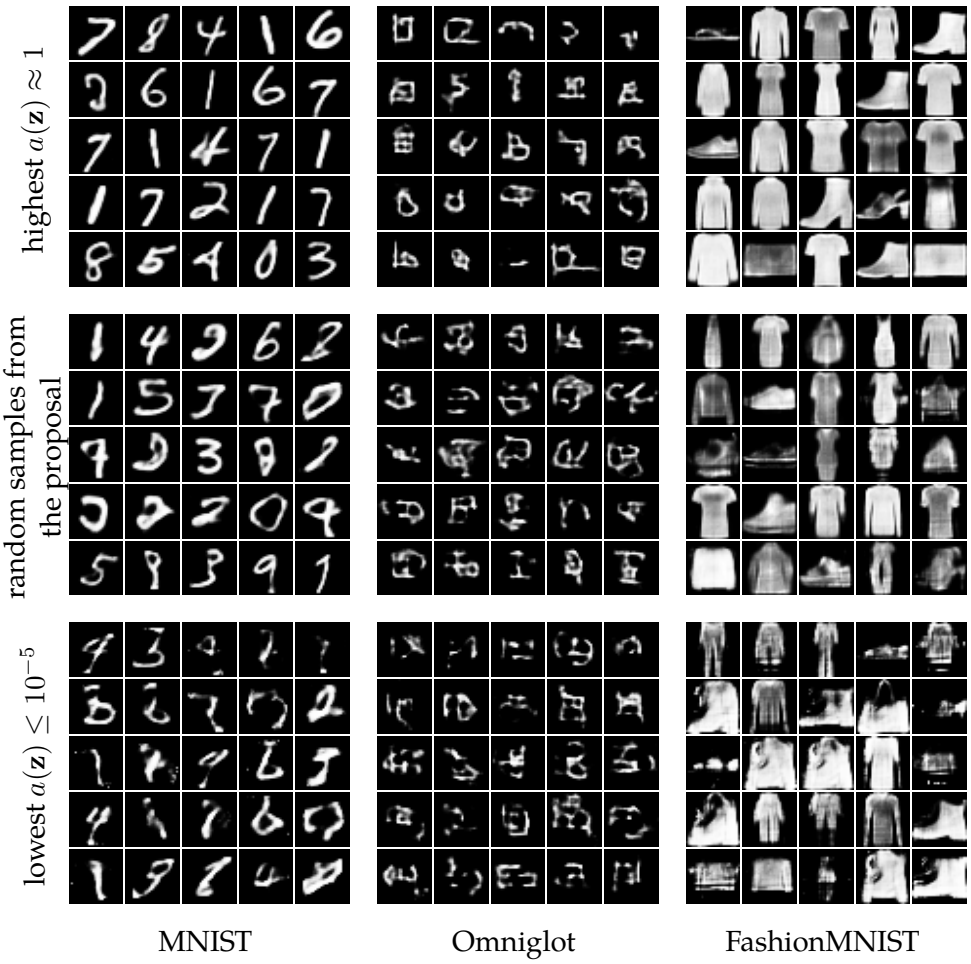


Figure 5.9: Ranked sample means *from the proposal* of a VAE ($L = 1$, MLP) with jointly trained resampled LARS prior. We drew 10^4 samples and show those with highest $a(\mathbf{z})$ (top) and lowest $a(\mathbf{z})$ (bottom)

Thus, our approach is able to (i) automatically detect good samples, and (ii) reject low-quality samples.

5.8.2 *Dimension-wise resampling*

Instead of resampling the entire vector $\mathbf{z}$, a less wasteful sampling scheme would be to resample each dimension independently according to its own learned acceptance function $a_i(z_i)$ (or in blocks of several dimension). However, such a factorised approach did not improve over a standard (factorised) Normal prior in our case. We speculate that it might be more effective when the target density has strong factorial structure, as might be the case for β-VAEs (Higgins et al., 2017); however, we leave this as future work.

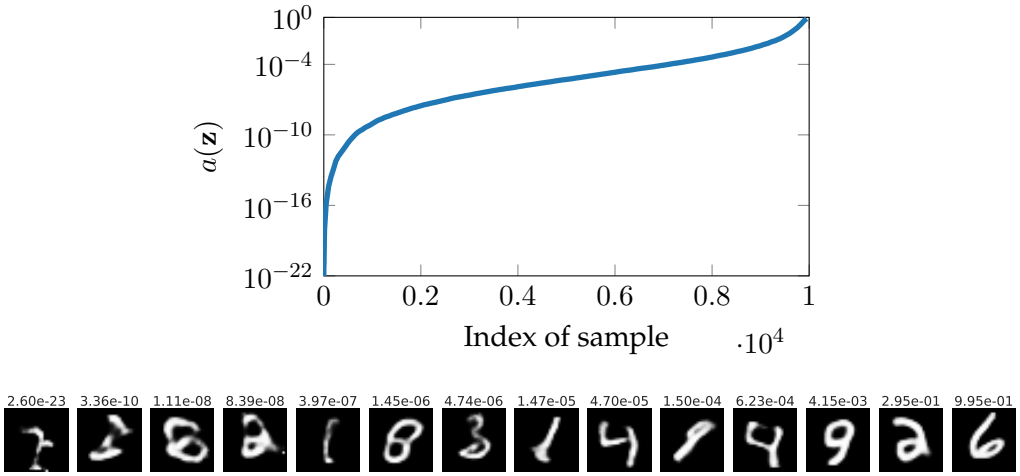


Figure 5.10: Ranked samples from the proposal of a jointly trained VAE model with resampled prior on MNIST and their corresponding acceptance function values. *top:* Distribution of the $a(\mathbf{z})$ values (index sorted by $a(\mathbf{z})$); note the log scale for $a(\mathbf{z})$. The average acceptance probability for this model is $Z \approx 0.014$. *bottom:* Representative samples and their $a(\mathbf{z})$ values. The $a(\mathbf{z})$ value correlates with sample quality.

5.8.3 *VAE with 2D latent space*

For illustrative purposes, we also trained a VAE with a two-dimensional latent space, $d_{\mathbf{z}} = 2$, with and without resampled priors. In Figure 5.1 we already presented the results when training with a standard Normal prior. In addition, we considered three different cases with resampled priors that use the standard Normal prior as proposal:

1. a VAE with a resampled prior with expressive/large network as acceptance function;
2. a VAE with a resampled prior with limited/small network as acceptance function; and
3. a pretrained VAE to which we apply LARS post-hoc; we fixed encoder/decoder and only trained the acceptance function.

Before inspecting the distributions in the latent space, we note that the NLL for these cases are all very similar. In fact, Hoffman and Johnson (2016) already show that the marginal KL gap is very small in this case ($d_{\mathbf{z}} = 2$).

REGULAR VAE. In Figure 5.12 we reproduce Figure 5.1 and show the aggregate posterior, that is, the mixture density of all encoded training data, $q(\mathbf{z}) = \frac{1}{N}\sum_n^N q(\mathbf{z} \mid \mathbf{x})$, as well as the standard Normal prior for a regular VAE with standard Normal prior. We find that the mismatch between standard Normal prior and aggregate posterior is $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z})) \approx 0.4$.

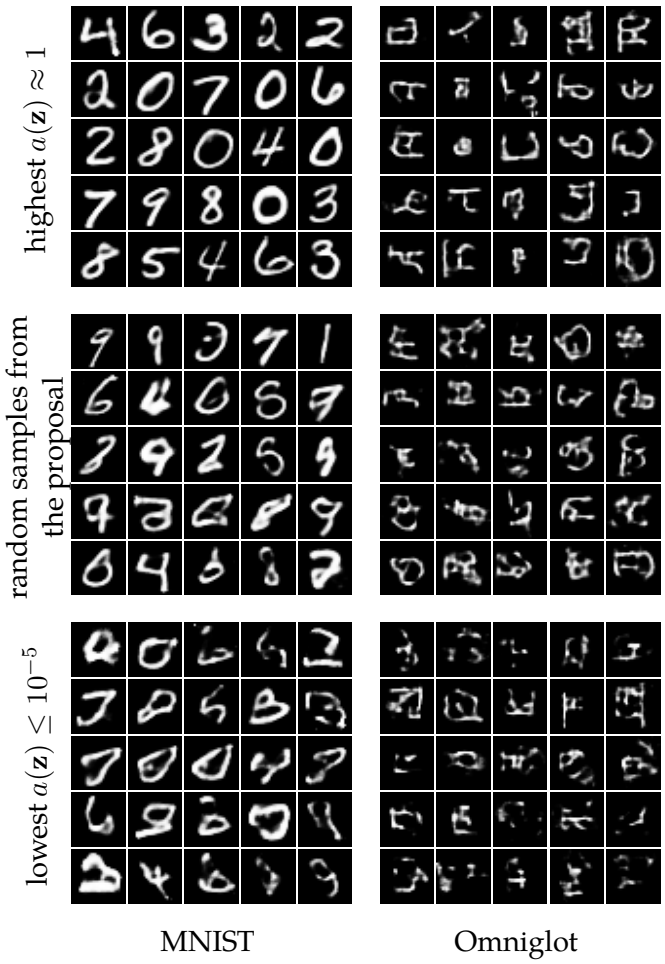


Figure 5.11: Sample means from a VAE with pretrained MLP encoder/decoder to which we applied a resampled LARS prior *post-hoc*. Samples shown are out of 10^4 samples drawn. *top:* samples with highest $a(\mathbf{z})$; *middle:* random samples from the proposal; *bottom:* samples with lowest $a(\mathbf{z})$.

VAE WITH JOINTLY TRAINED RESAMPLED PRIOR. In Figure 5.13 we show the aggregate posterior, proposal, accept function, and resampled density for the VAE with jointly trained resampled prior with high capacity/expressive accept function. We found that the LARS prior matches the aggregate posterior very well and has $\mathrm{KL}(q(\mathbf{z}) \| p(\mathbf{z})) \approx 0.15$. We note that the aggregate posterior is spread out more compared to the regular VAE, see also Figure 5.16 and note that the KL between the proposal and the aggregate posterior is larger than for the regular VAE. The accept function slices out parts of the prior very effectively and redistributes the weight towards the sides. In Figure 5.14 we plot the same as Figure 5.13 but with a smaller network for the acceptance function. We used the same random seed for both experiments and notice that the structure of the resulting latent space is very similar but more coarsely partitioned compared to the previous case. The final KL between aggregate posterior and resampled prior

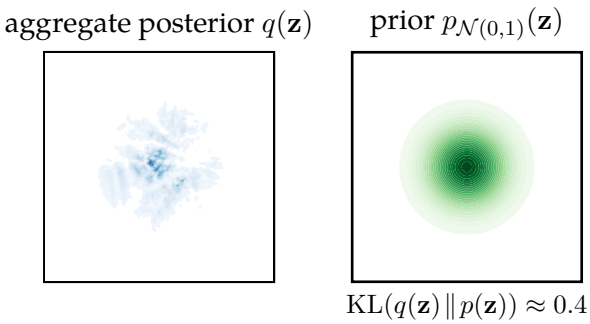


Figure 5.12: Aggregate posterior and fixed standard Normal prior for a regular VAE trained with the standard Normal prior on dynamic MNIST.

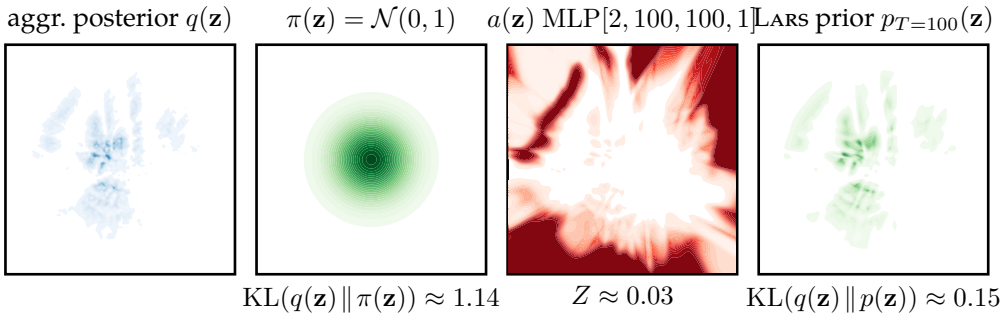


Figure 5.13: Aggregate posterior and resampled prior for a VAE with Lars prior and **high capacity** network for a: $a = \mathrm{MLP}[2 - 100 - 100 - 1]$. Dynamic MNIST.

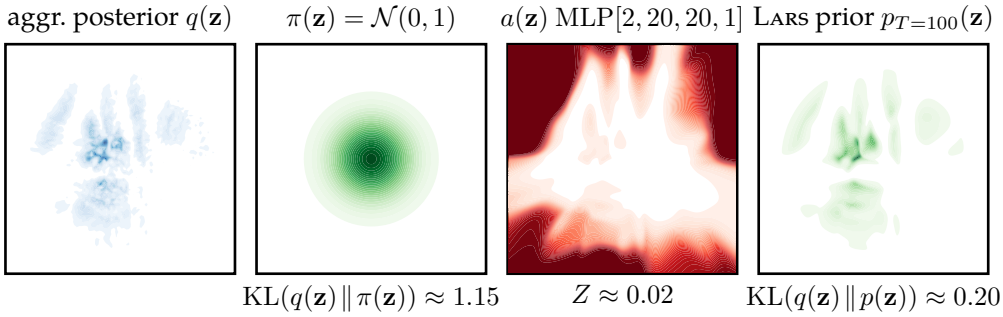


Figure 5.14: Aggregate posterior and resampled prior for a VAE with Lars prior and **low capacity** network for a: $a = \mathrm{MLP}[2 - 20 - 20 - 1]$. Dynamic MNIST.

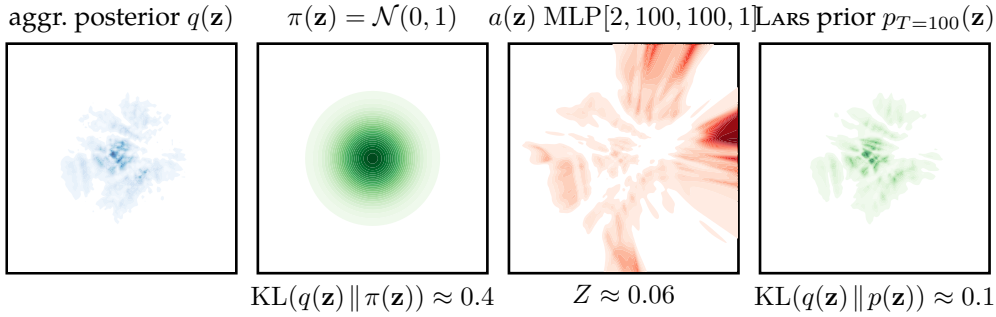
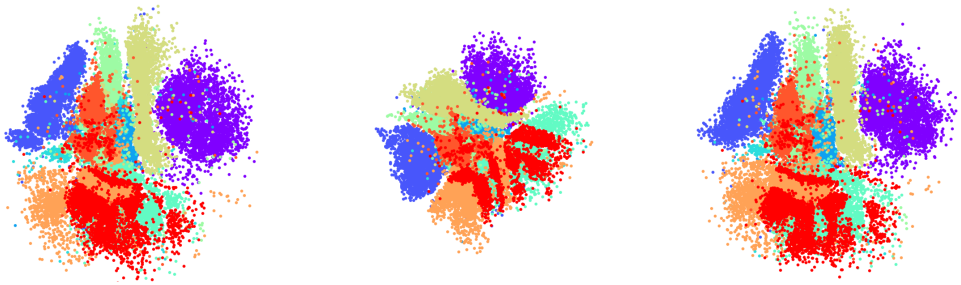


Figure 5.15: Aggregate posterior and resampled prior that has been trained *post-hoc* on the pretrained regular VAE. Dynamic MNIST.

is slightly larger, $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z})) \approx 0.20$, which we attribute to the lower flexibility of the a function.

RESAMPLED PRIOR FOR A PRETRAINED VAE We also consider the case of applying LARS post hoc to the pretrained regular VAE, see Figure 5.15, that is, we only learn a on a fixed encoder/decoder. Thus, the aggregate posterior is identical to Figure 5.12. However, we find that the acceptance function is able to modulate the standard Normal proposal to fit the aggregate posterior very well and even slightly better than in the jointly trained case. Note that the aggregate posterior is the same as in Figure 5.12 but that the LARS prior matches it much better than the standard Normal proposal.

In Figure 5.16 we compare scatter plots of the encoded means for all training data points for both the VAE with standard Normal prior as well as the LARS/resampled prior with large and small network for the acceptance function. Colours indicate the different MNIST classes.



VAE with LARS prior $a = \mathrm{MLP}[2, 100, 100, 1]$ | VAE with $\mathcal{N}(0, 1)$ prior | VAE with LARS prior $a = \mathrm{MLP}[2, 20, 20, 1]$

Figure 5.16: Embedding of the MNIST training data into the latent space of a VAE. Colours indicate the different classes and all plots have the same scale.

5.9 RESAMPLING DISCRETE OUTPUTS OF A VAE

So far, we have applied LARS only to the continuous variables of the relatively low-dimensional latent space of a VAE. However, our approach is more general and can also be used to resample *discrete and high-dimensional* random variables. We apply LARS directly to the 784-dimensional binarised (Bernoulli) outputs of a standard VAE in the pixel space. That is, we make the marginal likelihood richer by resampling it in the same way as the prior above: $\log p(\mathbf{x}) = \log \frac{\pi(\mathbf{x})a(\mathbf{x})}{Z}$, where $\pi(\mathbf{x})$ is now the original marginal likelihood, $a(\mathbf{x})$ is the acceptance function in

the pixel space, and Z is the normalisation constant. We can lower-bound the enriched marginal likelihood via its ELBO:

$$\mathbb{E}_{q_\varphi(\mathbf{z} \mid \mathbf{x})} \log \pi_\theta(\mathbf{x} \mid \mathbf{z}) - \mathrm{KL}(q_\varphi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})) + \log \frac{a_\lambda(\mathbf{x})}{Z}, \tag{5.43}$$

where $\mathbf{x}$ corresponds to the observation under consideration and we have reintroduced the subscripts φ, θ, λ to indicate the learnable parameters of the encoder, decoder, and acceptance function, respectively. The normaliser Z is estimated on decoded samples from the (regular) prior $p(\mathbf{z})$:

$$Z = \int \pi_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) a_\lambda(\mathbf{x}) \,\mathrm{d}\mathbf{x} \,\mathrm{d}\mathbf{z} \tag{5.44}$$

$$= \mathbb{E}_{\pi_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}[a_\lambda(\mathbf{x})] \approx \frac{1}{S} \sum_s^S a_\lambda(\mathbf{x}_s), \tag{5.45}$$

with $\mathbf{x}_s \sim \pi(\mathbf{x} \mid \mathbf{z}_s)$ and $\mathbf{z}_s \sim p(\mathbf{z})$. Note that because the images in the MNIST dataset are binary, both the observations $\mathbf{x}$ as well as the model samples $\mathbf{x}_s$ are discrete variables. We used the same MLP encoder/decoder as in our previous experiments, but increased the size of the $a(\mathbf{z})$ network to match the architecture of the encoder; we also employed truncated resampling with $T = 100$ as before. This does not make learning the parameters λ of the acceptance function $a_\lambda(\mathbf{x})$ more difficult than in the continuous case, as it does not involve propagating gradients through $\mathbf{x}$.

However, the situation is different for the parameters θ of the decoder, as the distribution of the model samples $\mathbf{x}_s$ and, thus, our estimate of Z depends on them. Therefore we need to compute the gradient of the sample-based estimate Z w.r.t. the decoder parameters θ; this would be be easy to do by using the reparameterisation trick if $\mathbf{x}_s$ were continuous. Unfortunately, in our case, the samples $\mathbf{x}_s$ are discrete and thus non-reparameterisible, which means that we must use a more general gradient estimator such as REINFORCE (Williams, 1992). Estimators of this type usually have much higher variance than reparameterisation gradients; for simplicity we choose to ignore this contribution to the gradients of the decoder parameters. Computationally, this amounts to estimating Z using

$$Z \approx \frac{1}{S} \sum_s^S a_\lambda(\texttt{stop\_grad}(\mathbf{x}_s)). \tag{5.46}$$

Thus, the encoder and decoder parameters are trained effectively by optimising the first two terms of Equation (5.43), whereas the parameters of the acceptance function are trained by optimising the last term of Equation (5.43). This is equivalent to the post-hoc setting considered in Figure 5.11 because the acceptance function has no effect on training the rest of the model.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	84.97	1
VAE + resampled $\mathcal{N}(0, 1)$	**83.04**	0.015
VAE with resampled output	83.63	0.014

Table 5.9: Test NLL on dynamic MNIST for a VAE with *resampled outputs*.

Resampling the outputs of a VAE is slower than resampling the prior for several reasons:

1. we need to decode all prior samples $\mathbf{z}_s$,
2. a larger $a(\mathbf{x})$ has to act on a higher dimensional space, and
3. estimation of Z typically requires more samples to be accurate (we used $S = 10^{11}$).

These are also the reasons why we apply LARS in the lower-dimensional latent space to enrich the prior in all other experiments in this paper.

Still, LARS works fairly well in this case. In terms of NLL, the VAE with resampled outputs outperforms the original VAE by over 1 nat (Table 5.9) and is only about 0.6 nats worse than the VAE with resampled prior. The learned acceptance function can be used to rank the binary output samples, see Figure 5.17.

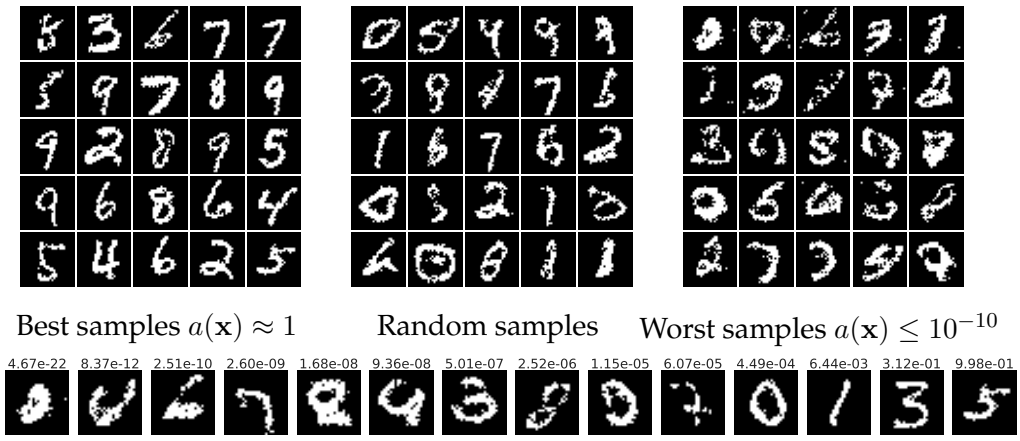


Figure 5.17: Samples from a VAE with a jointly trained acceptance function on the *output*, that is, with a resampled marginal likelihood. *top:* Samples grouped by the acceptance function (best, random, and worst); *bottom:* Samples and their acceptance function values.

5.10 COMPARISON AND RELATION TO ALTERNATIVE APPROACHES

Expressive prior distribution for VAEs.

Recently, several methods of improving VAEs by making their priors more expressive have been proposed. The VampPrior (Tomczak and Welling, 2018) is parameterised as a mixture of variational posteriors obtained by applying the encoder to a fixed number of learned pseudo-observations. While sampling from the VampPrior is fast, evaluating its density can be relatively expensive as the number of pseudo-observations used tends to be in the hundreds and the encoder needs to be applied on each one. A mixture of Gaussian prior (Nalisnick et al., 2016; Dilokthanakul et al., 2016) provides a simpler alternative to the VampPrior, but is harder to optimise and does not perform as well (Tomczak and Welling, 2018). Autoregressive models parameterised with neural networks allow specifying very expressive priors (e.g. Gregor et al., 2015; Gulrajani et al., 2017; Papamakarios et al., 2017) that support fast density evaluation but sampling from them is slow, as it is an inherently sequential process. While sampling in LARS also appears sequential, it can be easily parallelised by considering a large number of candidates in parallel. Chen et al. (2017) used autoregressive flows to define priors which are both fairly fast to evaluate and sample from. Flow-based approaches require invertible transformations with fast-to-compute Jacobians, see Section 1.6.2; these requirements limit the power of any single transformation and thus necessitate chaining a number of them together. In contrast, LARS places no restriction on the parameterisation of the acceptance function and can work well even with fairly small neural networks. On the other hand, evaluating models trained with LARS requires estimating the normalising constant using a large number of MC samples. However, as we showed above, LARS and flows can be combined to give rise to models with better objective value as well as higher sampling efficiency.

In a follow up work to LARS, Lawson et al. (2019) very recently introduced "energy inspired models"; they define the prior distribution through the sampling procedure rather than using a sampling procedure to approximate an energy based model as is effectively done by LARS. While their approach avoids the computation of a normaliser, it only provides a lower bound to the density. Moreover, energy inspired models can be extended to other sampling procedures, such as importance sampling or Hamiltonian Monte Carlo.

While not introduced as prior distribution for VAEs, the Gaussian Process Density Sampler (GPDS) (Murray et al., 2009) constitutes a similar approach for density modelling. A GPDS is fully non-parameteric and uses a squashed latent Gaussian process function for reweighting a proposal instead of a neural network.

It allows for Bayesian inference over potential sample histories that could have given rise to the observed dataset. Ultimately, Murray et al. (2009) only consider indefinite resampling whereas LARS also treats truncated rejection sampling.

Rejection sampling (RS).

Variational Rejection Sampling (Grover et al., 2018) uses RS to make the variational posterior closer to the exact posterior; as it relies on evaluating the target density, it is not applicable to our setting. Naesseth et al. (2017) derive reparameterisation gradients for some not directly reparameterisable distributions. They use RS to correct for sampling from a reparameterisable proposal instead of the distribution of interest. Similarly to VRS, this method requires being able to evaluate the target density.

Density ratio Estimation (DRE).

DRE (Sugiyama et al., 2012) estimates a ratio of densities by training a classifier to discriminate between samples from them. It has been used to estimate the KL term in the ELBO in order to train VAEs with implicit priors (Mescheder et al., 2017), but the approach tends to overestimate the ELBO (Rosca et al., 2018), making model comparison difficult. The acceptance probability function learned by LARS can be interpreted as estimating a (rescaled) density ratio between the aggregate posterior and the proposal, which is done end-to-end as a part of the generative process. Note that we do not aim to estimate the density ratio as accurately as possible. Instead, we aim to learn a compact model for $a(\mathbf{z})$, so that the resulting $p(\mathbf{z})$ strikes a good balance between approximating $q(\mathbf{z})$ well and generalising to new observations. Moreover, as the resampled prior has an explicit density, we can perform reliable model comparison by estimating the normalising constant using a large number of samples.

Products of Experts.

The density induced by LARS can be seen as a special instance of the product-of-experts (PoE, (Hinton, 2000)) architecture with two experts. However, while sampling from PoE models is generally difficult as it requires MCMC algorithms, our approach yields exact independent samples because of its close relationship to classical rejection sampling.

5.11 SUMMARY

We introduced Learned Accept/Reject Sampling (LARS), a powerful method for approximating a target density q given a proposal density and samples from the target. LARS uses a learned acceptance function to reweight the proposal, and can be applied to both continuous and discrete variables. We employed LARS to define a resampled prior for VAEs and showed that it outperforms a standard Normal prior by a considerable margin on several architectures and datasets. It is competitive with other approaches that enrich the posterior or prior and can be efficiently combined with the ones that support efficient sampling, such as flows. LARS can also be applied post hoc to already trained models and its learned acceptance function effectively ranks samples from the proposal by visual quality.

We addressed two challenges of our approach: potentially low sampling efficiency and the requirement to estimate the normalisation constant, and demonstrated that inference remains tractable and that a truncated resampling scheme provides an interpretable way to trade off sampling efficiency against approximation quality.

LARS is not limited to variational inference and exploring its potential for density modelling in other contexts is an interesting direction for future research. It is orthogonal to other recent approaches for density estimation, such as flow-based models, and holds several advantages but also disadvantages over them. On the one hand, LARS can model discontinuities – zero densities and hard cuts – that are notoriously difficult to model with continuous models. Moreover, it can equally be applied to continuous and discrete variables and training is very stable compared to, for example, deep flows. On the other hand, it involves estimation of an intractable normalisation constant and may yield a very low sampling efficiency when the mismatch between proposal and target are large. In the case of priors for VAEs we showed how these limitations can be mitigated. LARS' successful application to resampling the discrete outputs of a VAE is a first step in this direction.

We also envision generalising our approach to more than one acceptance function, for example resampling dimensions independently, or using a different resampling scheme for the first T steps and for the following attempts.

Part III

PROBABILISTIC INFERENCE IN FEW-SHOT LEARNING

6

INTRODUCTION TO FEW-SHOT LEARNING

So far, this thesis discussed supervised and unsupervised learning problems for which the i.i.d. assumption holds. We have assumed that the data for both training and testing set have been drawn independently and identically distributed from the same underlying data distribution. However, for many tasks this might not be the case; for instance, image properties or label classes might change in a classification task, or the properties of a dynamical system might get altered over time in a regression setting. While we would like our learning methods to be robust to these changes, most methods in machine learning, particularly statistical ones, fail under these conditions.

In contrast, a child encountering images of helicopters for the first time is able to generalise to instances with radically different appearance from only a handful of labelled examples. This remarkable feat is supported in part by a high-level feature-representation of images acquired from past experience. However, it is likely that information about previously learned concepts, such as aeroplanes and vehicles, is also leveraged; for example, sets of features like tails and rotors or objects like pilots and drivers are likely to appear in new images.

Moreover, many applications require predictions to be made on myriad small, but related datasets. In such cases, it is natural to desire learners that can rapidly adapt to new datasets at test time. These applications have given rise to significant interest in *few-shot learning* (Fei-Fei et al., 2006; Fink, 2005; Lake et al., 2011), which emphasises *data efficiency* via information sharing across related tasks. Despite recent advances, notably in meta-learning based approaches (Ravi and Larochelle, 2017; Vinyals et al., 2016; Edwards and Storkey, 2017; Finn et al., 2017; Lacoste et al., 2018), there remains a lack of general purpose methods for flexible, data-efficient learning.

The remainder of this thesis deals with few-shot learning, which is a particular instance of the domain shift introduced above. In discriminative few-shot learning we aim to classify instances based on very few examples or *shots* per class, in some cases as few as just one. In addition to these few examples, we have access to a large repository of examples from different but related classes. Our goal is to build machine learning systems for performing few-shot learning, which leverage both feature representations of the inputs and class information that have both been honed by learning from large amounts of labelled data in the repository.

This chapter provides an overview of few-shot learning and introduces two perspectives from which to approach it: transfer learning and meta-learning. In Chapters 7 and 8 we present two probabilistic frameworks in this vein.

6.1 TRANSFER LEARNING AND META-LEARNING

Since its inception, few-shot learning tasks have been addressed predominantly from two perspectives – transfer learning (Thrun, 1996) and meta-learning (Naik and Mammone, 1992; Thrun and Pratt, 2012; Schmidhuber, 1987). Both are established overlapping sub-fields of machine learning and data science and are, by themselves, much broader than few-shot learning; however, here we focus on aspects directly relevant to the few-shot learning task. See Pan and Yang (2010) for a broader survey on transfer learning and Lemke et al. (2015) and Vanschoren (2018) for recent surveys on meta-learning. In practice, a clear categorisation of an algorithm as either transfer learning or meta-learning is sometimes difficult, because many approaches contain aspects of both. And while there is no inherent benefit in labelling methods as belonging to one or the other, the two perspectives offer different approaches and strategies, each with their own merits. While both of them utilise some form of feature extractor, they differ in how the class information in the repository dataset is exploited. In the following we try to disambiguate the two.

Transfer learning

Transfer learning addresses how previously acquired knowledge and representations can be leveraged to improve performance on a new but related task; it emphasises the learning of features and common structure on the large repository and aims to transfer both to the new task. Both the feature representations as well as the structure learned from the repository are usually utilised more explicitly. This is often achieved through modelling with shared parameters; we can therefore say that transfer learning focuses more on modelling aspects of the few-shot learning task.

Meta-learning

Meta-learning instead emphasises meta-algorithms that can adapt a model or learner to new tasks and usually views the training repository as a collection of many smaller tasks that mimic the final evaluation task. Its central tenet is often summarised as "learning to learn" (Thrun and Pratt, 2012). In other words, rather than utilising the class information to extract structure explicitly, the repository serves as a testbed to learn how to adapt or fit a model to new tasks. We can therefore say that meta-learning focuses more on inference aspects of the few-shot learning task. Minka and Picard (1997) discuss "learning to learn" from the perspective of i.i.d. sampling over *sets of points* (or tasks). This view naturally gives rise to episodic training discussed below.

Multi-task learning

Multi-task learning is related and partly subsumed both within transfer but also

meta-learning. It aims to train a system that performs well at several related tasks as once (Caruana, 1997; Ruder, 2017). It naturally generalises to transfer learning or meta-learning when new classes, or more generally tasks, are added at test time.

In practice, transfer learning approaches are often conceptually simpler and easier to apply than many meta-learning approaches. As we discuss further below, the former usually rely on strong pre-trained feature extractors and are less adaptable than meta-learning approaches, which can often be applied to a larger variety of settings.

6.2 THE FEW-SHOT LEARNING TASK

In few-shot learning we generally distinguish between two phases of learning whose names are inspired by the meta-learning perspective (Ravi and Larochelle, 2017) but also apply to transfer learning: (i) meta-training and (ii) meta-testing.

6.2.1 *The two phases of few-shot learning*

meta-training

In the *meta-training phase* we receive a large dataset or repository $\widetilde{\mathcal{D}} = \{\widetilde{\mathbf{x}}_i, \widetilde{y}_i\}_{i=1}^{\widetilde{N}}$ of images $\widetilde{\mathbf{x}}_i$ and corresponding labels $\widetilde{y}_i \in \{1, \dots, \widetilde{C}\}$ that indicate which of the $\widetilde{C}$ classes each image belongs to. Typically, the dataset contains many instances per class. In some cases, we might not access or have access to the entire large dataset at once but only subsampled *tasks* from it. We will elaborate on this below when addressing episodic training particular to meta-learning.

meta-testing

In the *meta-testing phase*, we receive a small dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{C \cdot k}$ of C *new* classes, $y_i \in \{\widetilde{C} + 1, \widetilde{C} + C\}$, with k images from each new class. Typically, k is very small and on the order of 1 to 20. We correspondingly refer to the few-shot learning task as a C-way-k-shot task, because we ultimately distinguish between C classes with k labelled example instances per class.

Regardless of perspective, our goal is to construct a model that can leverage the information in $\widetilde{\mathcal{D}}$ and $\mathcal{D}$ to predict well on unseen images $\mathbf{x}^*$ from the new classes; the performance is evaluated on prediction accuracy against ground truth labels y^* from the new classes.

In addition to this simple few-shot learning task there exist many possible extensions, such as online- or continual learning, where the set of new classes grows over time, both in terms of novel classes as well as examples per class, or we also include the repository classes in the evaluation.

The different transfer learning and meta-learning approaches to this task utilise the provided data in different ways. Broadly speaking, there are two main

strategies to incorporate the large dataset $\widetilde{\mathcal{D}}$ that are roughly aligned with the division between transfer and meta-learning.

6.2.2 *The transfer learning perspective*

Consider the problem from a transfer learning perspective and use the large dataset $\widetilde{\mathcal{D}}$ all at once, for example to pre-train a feature extractor and then transfer these features to the meta-testing classes in $\mathcal{D}$, see Figure 6.1. This approach still requires a structure to actually perform the feature transfer. For example, a simple approach is to train a generalised logistic regression using the pre-trained features from the few-shot examples $\mathcal{D}$ at meta-test time; when enough meta-test data is available, the features can also be fine-tuned.

In Chapter 7 we utilise this perspective and develop a simple probabilistic model to perform the transfer of features. Because we use the large repository all at once, these approaches can typically contain powerful feature extractors, such as VGG (Simonyan and Zisserman, 2015) or ResNet (He et al., 2016) for images.

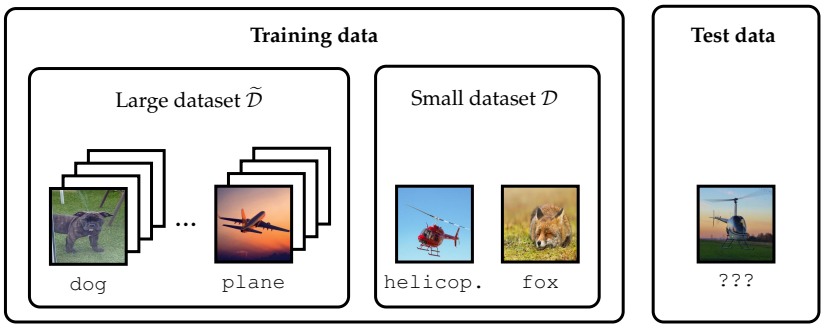


Figure 6.1: The few-shot learning task from a transfer-learning perspective. During meta-training we use the large dataset $\widetilde{\mathcal{D}}$ to train a learning system. At meta-test time we use the small dataset $\mathcal{D}$ to adapt the system and then test on the test data. Here we illustrate a 2-way-1-shot task. Images from the ImageNet dataset (Russakovsky et al., 2015).

6.2.3 *The meta-learning perspective*

episodic training

Consider the problem from a meta-learning perspective and use the large dataset to perform *episodic training*, in which we simulate many small few-shot tasks by sub-sampling classes and training/testing images from $\widetilde{\mathcal{D}}$, see Figure 6.2. In this approach we typically train a meta-algorithm that performs the adaptation to the new task at hand; still, part of the model architecture functions as a form of feature extractor, whose features are effectively transferred from the large dataset $\widetilde{\mathcal{D}}$. Note that both during meta-training and meta-testing the data per task are

split into a train and a test set, giving rise to "meta-train train/test splits" and "meta-test train/test splits".

Due to the nature of episodic training, it is often difficult to train powerful feature extractors in this setting. Moreover, as we discuss below, some approaches are architecturally constrained or only perform well on the same C-way-k-shot task they have been trained on episodically.

In Chapter 8 we follow the meta-learning perspective and present an approach to meta-learn probabilistic inference for prediction.

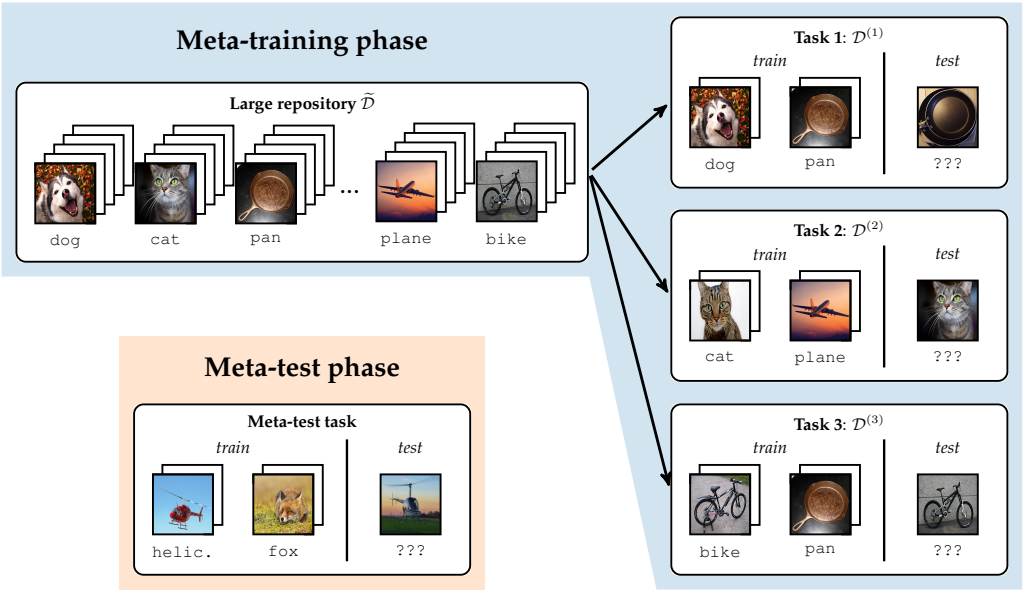


Figure 6.2: The few-shot learning task from a meta-learning perspective. During the meta-training phase, we sample many small tasks t with datasets $\mathcal{D}^{(t)}$ from the large repository $\widetilde{\mathcal{D}}$ to train the meta-learning system episodically. In the meta-test phase, we evaluate the system on a test task with new classes. Here we illustrate a 2-way-2-shot task. Images from the ImageNet dataset (Russakovsky et al., 2015).

6.3 OUR APPROACHES TO PROBABILISTIC FEW-SHOT LEARNING

In the next two chapters, we present two probabilistic frameworks for few-shot learning. Both build on top of earlier hierarchical probablistic models for multi-task and transfer learning (Heskes, 2000; Bakker and Heskes, 2003) and emphasise data-efficient learning.

Our approach in Chapter 7 follows the transfer learning perspective and uses simple probabilistic models for the adaptive heads of a deep classifier for the transfer. It employs a powerful feature extractor that is pre-trained on the large repository, and a special case of this approach can be interpreted as automatically regularised softmax regression.

In contrast, our approach in Chapter 8 follows the meta-learning perspective and proposes to meta-learn approximate probabilistic inference for prediction (ML-PIP) using amortised head models. It employs simpler feature extractors that are jointly trained with the amortisation network in an episodic fashion. Compared to the previous approach it does not require test-time optimisation and can be more generally applied, for example to few-shot regression tasks.

We provide more comparisons below and in Chapters 7 and 8, where we introduce the approaches in more detail.

6.4 RECENT ADVANCED IN FEW-SHOT LEARNING

Few-shot learning has enjoyed a recent resurgence in the academic community. This section summarises some of the recent advances from the literature which we also compare to in Chapters 7 and 8. We separate them into three rough categories: (i) metric-based methods, (ii) gradient-based methods, and (iii) deep probabilistic methods.

Our transfer learning approach in Chapter 7 belongs to the category of deep probabilistic methods, whereas ML-PIP (Chapter 8) does not belong to either of them. It is most closely related to the recently proposed Neural Processes (Garnelo et al., 2018a) for conditional modelling as we discuss in Section 8.3. In fact, there we also show that many of the methods discussed here can be derived as special cases of ML-PIP.

For more comprehensive surveys on few-shot learning, including methods not addressed here, see Chen et al. (2019) and Wang and Yao (2019).

6.4.1 *Metric-based methods*

Metric-based methods map the few-shot training and test points into a non-linear embedding space and perform classification by assessing which training points are closest, according to a metric, to the test points. Both the embedding (feature representation) as well as the metric can be learnable, and the methods mostly differ in these two aspects. These approaches are typically viewed as meta-learning approaches because they are trained episodically and the learned metrics can be viewed as adaptation at test time.

Siamese Networks

Siamese Networks (Koch et al., 2015) train the embedding using a same/different prediction task between pairs of images derived from the repository dataset and use a weighted L_1 metric for classification. In essence, the network predicts the probability that the two input images belong to the same class. To classify a new test point $\mathbf{x}^*$ from the meta-testing test set, Koch et al. (2015) compute

the pairwise similarities between it and the available meta-test training points $\mathcal{S} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N=k\cdot C}$ (also referred to as support set); $\mathbf{x}^*$ is classified according to the closest $\mathbf{x}_n$.

Matching Networks

Matching Networks (Vinyals et al., 2016) episodically train an embedding defined through an attention mechanism $a(\cdot, \cdot)$ that linearly combines training labels weighted by their proximity to test points:

$$p(y^* \mid \mathbf{x}^*, \mathcal{S}) = \sum_{n=1}^{N=k\cdot C} a(\mathbf{x}^*, \mathbf{x}_n)\, y_n \,, \tag{6.1}$$

where $\mathcal{S}$ denotes the support set, and the attention mechanism $a(\cdot, \cdot)$ expresses the weight (similarity to $\mathbf{x}^*$) of each label, similar to kernel density estimation. Vinyals et al. (2016) use the cosine similarity between embeddings as attention mechanism:

$$a(\mathbf{x}^*, \mathbf{x}_n) = \frac{\exp(d_{\cos}(f(\mathbf{x}^*), g(\mathbf{x}_n)))}{\sum_{n'=1}^{N=k\cdot C} \exp(d_{\cos}(f(\mathbf{x}^*), g(\mathbf{x}_{n'})))} \tag{6.2}$$

where f and g are learned embeddings (feature representations), and $d_{\cos}$ denotes the cosine similarity.

Prototypical Networks

The more recently proposed Prototypical Networks (Snell et al., 2017) are a streamlined version of Matching Networks in which the embedded classes are summarised by their mean in the embedding space. A new example $\mathbf{x}^*$ is then only compared against these class means rather than all points in the support set:

$$\begin{aligned} p(y^* = c \mid \mathbf{x}^*, \mathcal{S}) &= \frac{\exp(-d(f(\mathbf{x}^*), \boldsymbol{\mu}_c))}{\sum_{c'=1}^{C} \exp(-d(f(\mathbf{x}^*), \boldsymbol{\mu}_{c'}))} \\ \boldsymbol{\mu}_c &= \frac{1}{|\mathcal{S}_c|} \sum_{\mathbf{x}_c \in \mathcal{S}_c} f(\mathbf{x}_c) \,, \end{aligned} \tag{6.3}$$

where $\mathcal{S}_c$ is the support set for class c, d is a learnable distance function, and f is a learnable embedding.

Recent efforts have improved the performance of Matching Networks and Prototypical Networks using data hallucination (Hariharan and Girshick, 2017; Wang et al., 2018) by training them jointly with a generative model of the data. These embedding methods learn representations for few-shot learning, but do not directly leverage concept transfer.

Relation Nets

Similarly to Siamese Networks, Relation Nets (Sung et al., 2018) score the similarity (referred to as relation score) of an embedded test example to embedded points in the support set and classify according to the point with highest similarity.

Instead of a weighted L_1 distance they express the similarity with a second network g on concatenated embeddings f:

$$r_{ij} = g(\texttt{concatenate}(f(\mathbf{x}_i), f(\mathbf{x}_j))) \tag{6.4}$$

where r_{ij} denotes the relation score between two points $\mathbf{x}_i$ and $\mathbf{x}_j$.

6.4.2 *Gradient-based methods*

Gradient-based methods adapt the parameters θ of a learner f_θ to a new task by performing one or several gradient updates with a specified loss function. These methods are also typically trained episodically and classified as meta-learning approaches. They differ in how the updates are performed.

LSTM meta-learner

In LSTM meta-learner (Ravi and Larochelle, 2017) the initialisation and parameter updates for the learner network are provided by a second network, the so-called meta-learner which is realised through an LSTM (Hochreiter and Schmidhuber, 1997).

MAML

Model agnostic meta learning (MAML) (Finn et al., 2017) performs a small number of gradient updates (often one) of the learner parameters θ from an initialisation θ^* using the loss on the meta-train training data to perform well on the corresponding test set. The initialisation θ^* is then learned over a distribution of tasks such that this task-specific fine-tuning through a single gradient step is most efficient. Note that this requires backpropagation through the gradient updates, which entails the evaluation of second order derivatives, though Finn et al. (2017) also present a first order approximation.

Many variations and extensions of MAML now exist, such as Bayesian MAML (Yoon et al., 2018) or Probabilistic MAML (Finn et al., 2018).

Reptile

Reptile (Nichol and Schulman, 2018) is similar in spirit to MAML; however, instead of one gradient update it performs several gradient update steps per task and then moves the parameters of the learner towards those parameters.

6.4.3 *Deep probabilistic methods*

Deep probabilistic methods include the approach developed in Chapter 7. The methods in this family are not unique to deep learning, and the idea of probabilistic modelling of weights has been applied in multi-task learning (Bakker and Heskes, 2003).

The work most closely related to our own in Chapter 7 is not an approach to few-shot learning *per se*, but rather a method for training CNNs with highly imbalanced classes (Srivastava and Salakhutdinov, 2013). It is similar in that it trains a form of Gaussian mixture model over the final layer weights using MAP inference that regularises learning.

Burgess et al. (2016) propose an elegant approach to few-shot learning that is an instance of the framework described in Chapter 7: a Gaussian model is fit to the weights with MAP inference. The evaluation is promising, but preliminary, and our approach provides a comprehensive evaluation.

While not using a probabilistic approach, Qiao et al., 2018 develop a method for few-shot learning that trains a recognition model to amortise MAP inference for the softmax weights, which can then be used at few-shot learning time. While this method trains the mapping from activation to weights jointly with the classifier, and thus does not learn from the weights *per se*, it does exploit the structure in the weights for few-shot learning.

7

A PROBABILISTIC TRANSFER APPROACH TO FEW-SHOT LEARNING

In this chapter we introduce a simple probabilistic approach for discriminative few-shot learning inspired by the transfer learning perspective in Chapter 6. It constitutes a general framework based on the combination of a deep feature extractor, trained on batch classification, and traditional probabilistic modelling.

Our approach not only leverages the feature-based representation learned by a neural network from the initial task (representational transfer), but also information about the classes (concept transfer). The concept information is encapsulated in a probabilistic model for the classifier weights of the neural network, which acts as a prior for probabilistic few-shot learning. We show that even a simple probabilistic model achieves state-of-the-art performance on a standard few-shot learning dataset by a large margin. Moreover, it is able to accurately model uncertainty, leading to well calibrated classifiers, and is easily extensible and flexible, unlike many recent approaches to few-shot learning.

Our framework subsumes two existing approaches in this vein (Srivastava and Salakhutdinov, 2013; Burgess et al., 2016), and is motivated by similar ideas from multi-task learning (Bakker and Heskes, 2003).

The research in this chapter is joint work with my co-first author Mateo Rojas-Carulla as well as Jakub Świątkowski, Bernhard Schölkopf, and Richard Turner. It was originally published as 'Discriminative k-shot learning using probabilistic models' (Bauer*, Rojas-Carulla* et al., 2017b). My main contributions were the joint development of the initial idea, the probabilistic model on the weights, as well as design and partial execution of the experiments.

7.1 A FRAMEWORK FOR PROBABILISTIC FEW-SHOT LEARNING

The intuition behind our approach is that deep learning will learn powerful feature representations, whereas probabilistic inference will transfer top-down conceptual information from old classes.

Representational learning is driven by the large number of training examples from the repository of original classes, making it amenable to standard deep learning with a convolutional neural network (CNN). This learns a rich repres-

entation of images at the top hidden layer of the CNN. Accumulated knowledge about classes is embodied in the top layer softmax weights of the network.

In contrast, the transfer of conceptual information to the new classes relies on a relatively small number of existing classes and few-shot data points, which means probabilistic inference is appropriate. The accumulated knowledge is extracted by training a probabilistic model on these weights; in other words, the softmax weights are effectively used as data to inform likely new weights at meta-test time using a probabilistic model. Few-shot learning can thus (i) use the representation of images provided by the CNN as input to a new softmax function, and (ii) learn the new softmax weights by combining prior information about their likely form derived from the repository dataset with the few-shot likelihoods at meta-test time.

In Figure 7.1 we illustrate the structure of the top level (classifier) weights of a VGG style network trained to classify CIFAR-100 images.

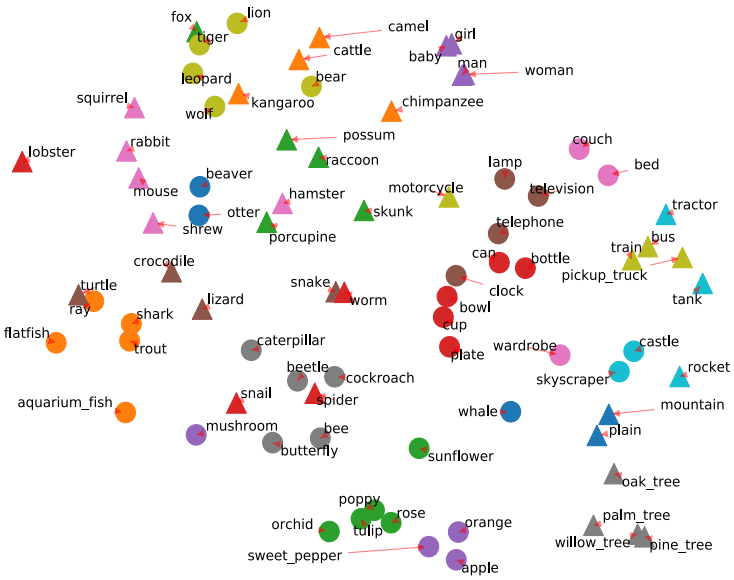


Figure 7.1: t-SNE embedding of the CIFAR-100 weights $\widetilde{\mathrm{W}}$ trained using a VGG style architecture. The points are coloured according to their respective superclass. The colouring by superclass makes the structure in the weights evident, as t-SNE overall recovers the structure in the dataset. For instance, oak tree, palm tree, willow tree and pine tree form a cluster on the bottom right. This structure motivates our approach, as the training weights contain information which may be useful at few-shot time, for instance given a few example from chestnut trees.

Following the transfer perspective we propose to use the large dataset $\widetilde{\mathcal{D}}$ to train a powerful feature extractor on batch classification, which can then be used in conjunction with a simple probabilistic model to perform few-shot learning. Bakker and Heskes (2003) introduced a general probabilistic framework for multi-task learning with multi-head models, in which all parameters of a generic

feature extractor are shared between a set of tasks, and only the weights of the top linear layer (the "heads") are task dependent. In the following, we frame few-shot learning in a similar setting and propose a probabilistic framework for few-shot learning in this vein. Our probabilistic framework comprises four phases that we refer to as 1) representational learning, 2) concept learning, 3) few-shot learning, and 4) few-shot testing (see Figure 7.3).

7.1.1 *Feature extractor and representational learning*

We first introduce a convolutional neural network (CNN) Φ_φ as feature extractor whose last hidden layer activations are mapped to two sets of softmax output units, corresponding to the $\widetilde{C}$ classes in the large repository dataset $\widetilde{\mathcal{D}} = \{\widetilde{\mathbf{x}}_n, \widetilde{y}_n\}_{n=1}^{\widetilde{N}}$ and the C classes in the small meta-test dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^{N=C\cdot k}$, respectively. These separate mappings are parametrised by weight matrices $\widetilde{\mathrm{W}}$ for the old classes and W for the new classes. Denoting the output of the final hidden layer as $\Phi_\varphi(\mathbf{x})$, the first softmax units compute $p(\widetilde{y}_n \mid \Phi_\varphi(\widetilde{\mathbf{x}})_n, \widetilde{\mathrm{W}}) = \mathrm{softmax}(\widetilde{\mathrm{W}}\Phi_\varphi(\widetilde{\mathbf{x}})_n)$ and the second $p(y_n \mid \Phi_\varphi(\mathbf{x})_n, \mathrm{W}) = \mathrm{softmax}(\mathrm{W}\Phi_\varphi(\mathbf{x})_n)$, see Figure 7.2. The weight matrices have shape $\widetilde{C} \times d_\Phi$ and $C \times d_\Phi$, respectively, where d_Φ denotes the dimensionality of the feature representation $\Phi_\varphi(\mathbf{x})$. Each row contains the weights that are associated with a particular class.

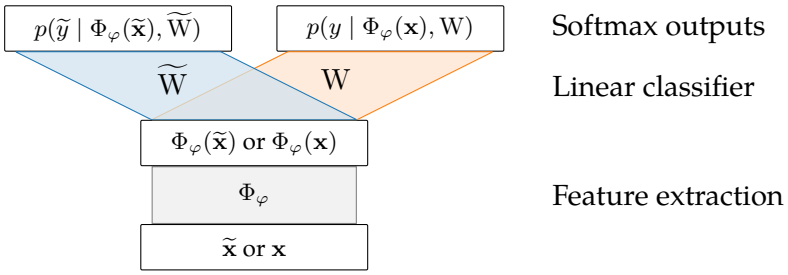


Figure 7.2: Two-head model for probabilistic few-shot learning.

For *representational learning (phase 1)* the repository dataset $\widetilde{\mathcal{D}}$ is used to train the CNN Φ_φ using standard deep learning optimisation approaches. This involves learning the parameters φ of the feature extractor up to the last hidden layer, as well as the softmax weights $\widetilde{\mathrm{W}}$. The network parameters φ are fixed from this point on and shared across later phases.

7.1.2 *Probabilistic modelling of the weights*

The next goal is to build a probabilistic method for few-shot prediction that transfers structure from the trained softmax weights $\widetilde{\mathrm{W}}$ to the new few-shot softmax weights W, and combines it with the few-shot training examples. Thus,

given a test image $\mathbf{x}^*$ during *few-shot testing (phase 4)*, we compute its feature representation $\Phi(\mathbf{x}^*)$, and the prediction for the new label y^* is found by averaging the softmax outputs $p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathrm{W})$ over the posterior distribution of the softmax weights given the two datasets $p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}})$,

$$p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathcal{D}, \widetilde{\mathcal{D}}) = \int p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathrm{W}) p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}}) \, \mathrm{dW}. \tag{7.1}$$

We therefore need to perform probabilistic inference on the softmax weights W given the large dataset $\widetilde{\mathcal{D}}$ and the small dataset $\mathcal{D}$. This is exactly what we do in the *concept learning phase (phase 2)* and the *few-shot learning phase (phase 3)*, respectively.

Probabilistic graphical model

To this end, we consider a general class of probabilistic models for the softmax weights, see Figure 7.3. We assume that the two sets of softmax weights are generated from shared hyperparameters θ, such that

$$p(\widetilde{\mathrm{W}}, \mathrm{W}, \theta) = p(\theta) p(\widetilde{\mathrm{W}} \mid \theta) p(\mathrm{W} \mid \theta) \tag{7.2}$$

as indicated in the graphical model in Figure 7.3. In this way, the large dataset $\widetilde{\mathcal{D}}$

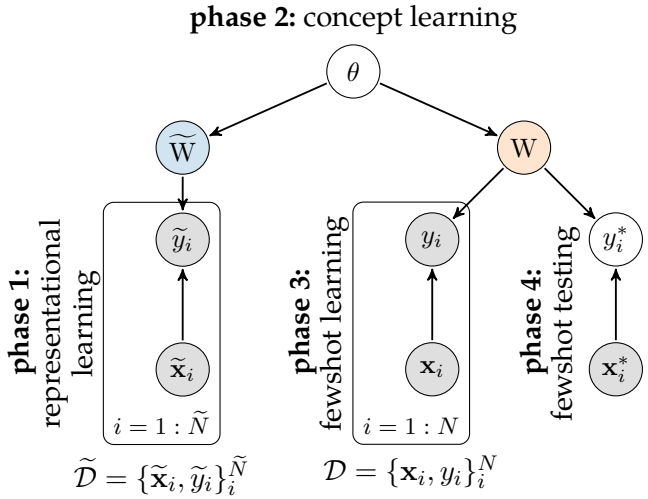


Figure 7.3: Graphical model for probabilistic few-shot learning.

contains information about θ that in turn constrains the new softmax weights W; in the concept learning phase, the initial dataset is used to form this posterior distribution over the concept hyperparameters $p(\theta \mid \widetilde{\mathcal{D}})$. The few-shot learning phase then combines the information about the new weights provided by $\widetilde{\mathcal{D}}$ with the information in the few-shot dataset $\mathcal{D}$ to form the posterior distribution $p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}})$.

In the following we explain the steps and necessary approximations to derive an expression for this posterior distribution in more detail and derive expressions for the corresponding final predictive distribution Equation (7.1).

We can break up these steps into concept learning (phase 2) and fewshot learning (phase 3)

$$p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}}) \propto \underbrace{p(\mathrm{W} \mid \widetilde{\mathcal{D}})}_{\text{concept learning}} \times \underbrace{p(\mathcal{D} \mid \mathrm{W})}_{\text{fewshot learning}}, \tag{7.3}$$

where

$$p(\mathrm{W} \mid \widetilde{\mathcal{D}}) = \int p(\mathrm{W} \mid \theta) p(\theta \mid \widetilde{\mathcal{D}}) \, \mathrm{d}\theta.$$

To see this, notice that

$$p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}}) \propto p(\mathrm{W}, \mathcal{D}, \widetilde{\mathcal{D}}) = p(\widetilde{\mathcal{D}}) p(W \mid \widetilde{\mathcal{D}}) p(\mathcal{D} \mid \widetilde{\mathcal{D}}, \mathrm{W}). \tag{7.4}$$

The graphical model in Figure 7.3 entails that $\mathcal{D}$ is conditionally independent from $\widetilde{\mathcal{D}}$ given W, such that

$$p(\mathcal{D} \mid \mathrm{W}, \widetilde{\mathcal{D}}) = p(\mathcal{D} \mid \mathrm{W}) \tag{7.5}$$

We recover Equation (7.3) by adding $p(\widetilde{\mathcal{D}})$ to the constant of proportionality.

Concept learning (phase 2)

Inference in this model is generally intractable and requires approximations. The main challenge is computing the posterior distribution over the hyperparameters given the initial dataset. However, progress can be made if we assume that the posterior distribution over the weights can be well approximated by the MAP value $p(\widetilde{\mathrm{W}} \mid \widetilde{\mathcal{D}}) \approx \delta(\widetilde{\mathrm{W}} - \widetilde{\mathrm{W}}^{\mathrm{MAP}})$. This is arguably a justifiable assumption as the initial dataset is large and so the posterior will concentrate on narrow modes (with similar predictive performance). In this case $p(\theta \mid \widetilde{\mathcal{D}}) \approx p(\theta \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}})$ and, as a consequence of this approximation and the structure of the model, the original data $\widetilde{\mathcal{D}}$ are not required for the few-shot learning phase. Instead, the weights learned from these data, $\widetilde{\mathrm{W}}^{\mathrm{MAP}}$, can themselves be treated as observed data, which induce a predictive distribution over the few-shot weights $p(\mathrm{W} \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}})$ via Bayes' rule.

Few-shot learning (phase 3)

During few-shot learning (phase 3) we treat this predictive distribution as our new prior on the weights and again use Bayes' rule to combine it with the softmax

likelihood of the few-shot training examples $\mathcal{D}$ to obtain an updated posterior over the weights that now also incorporates $\mathcal{D}$,

$$\begin{aligned} p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}}) &\approx p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \\ &\propto p(\mathrm{W} \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \prod_{n=1}^{N} p(y_n \mid \Phi_\varphi(\mathbf{x}_n), \mathrm{W}), \end{aligned} \tag{7.6}$$

where we have plugged in the data likelihood

$$p(\mathcal{D} \mid \mathrm{W}) = \prod_n p(y_n \mid \Phi_\varphi(\mathbf{x}_n), \mathrm{W}) \tag{7.7}$$

The posterior in Equation (7.6) is generally intractable. It can, however, be approximated using either its MAP estimate

$$p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \approx \delta(\mathrm{W} - \mathrm{W}^{\mathrm{MAP}}) \tag{7.8}$$

or through sampling

$$\mathrm{W}_m \sim p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathrm{W}}^{\mathrm{MAP}}). \tag{7.9}$$

Note that the posterior $p(\mathrm{W} \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}}) = \int p(\mathrm{W} \mid \theta) p(\theta \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \, \mathrm{d}\theta$ can also be analytic. This is the case for conjugate models or when a point estimate for θ is provided by the concept modelling stage instead of a full distribution; in the latter case we find that $p(\mathrm{W} \mid \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \approx p(\mathrm{W} \mid \theta^{\mathrm{MAP}})$.

Posterior predictive distributions

With these results for the posterior distribution over the weights, we can now approximate the posterior predictive distribution (Equation (7.1)). If we approximate Equation (7.6) by its MAP estimate $\mathrm{W}^{\mathrm{MAP}}$, the integral in Equation (7.1) becomes

$$\begin{aligned} p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathcal{D}, \widetilde{\mathcal{D}}) &\approx p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathcal{D}, \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \\ &\approx p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathrm{W}^{\mathrm{MAP}}). \end{aligned} \tag{7.10}$$

If we perform inference through sampling, the posterior predictive can be estimated as

$$p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathcal{D}, \widetilde{\mathrm{W}}^{\mathrm{MAP}}) \approx \frac{1}{M} \sum_{m=1}^{M} p(y^* \mid \Phi_\varphi(\mathbf{x}^*), \mathrm{W}_m). \tag{7.11}$$

7.2 CHOOSING A MODEL FOR THE WEIGHTS

The probabilistic model over the weights is key: a good model with the right inductive biases will transfer useful knowledge that improves performance. However, the usual trade-off between model complexity and learnability is particularly egregious in our setting as the weights $\widetilde{\mathrm{W}}$ are few and high-dimensional and the number of few-shot samples is small. To emphasise, each distinct class corresponds to a single "data point", and we only have $\widetilde{C}$ many data points to fit the probabilistic model; at the same time, the dimensionality of each weight vector is determined by the size of the last hidden layer, which can be very large in practice.

Here we present three different priors on the weights: (i) Gaussian models, (ii) Gaussian mixture models, and (iii) Laplace distributions. After extensive testing, we found that for the datasets considered, a Gaussian model for the weights strikes the best trade-off between complexity and learnability. A Gaussian model is therefore the main method we propose, see Section 7.3.2 for the detailed experimental model comparison.

7.2.1 *The context independence assumption*

Before introducing the particular models we make two simplifying assumptions. First, treating the weights from the hidden layer to the softmax outputs as a vector, we assume independence. Second, we assume the distribution between the weights of old and new classes to be identical,

$$p(\widetilde{\mathrm{W}}, \mathrm{W}, \theta) = p(\theta) \prod_{\widetilde{c}=1}^{\widetilde{C}} p(\widetilde{\mathbf{w}}_{\widetilde{c}} \mid \theta) \prod_{c=1}^{C} p(\mathbf{w}_c \mid \theta), \tag{7.12}$$

where $p(\widetilde{\mathbf{w}}_{\widetilde{c}} \mid \theta) \overset{\text{dist}}{=} p(\mathbf{w}_c \mid \theta)$. We refer to this assumption as *context independence assumption*, and also employ it in Chapter 8, where we present a meta-learning approach to probabilistic inference.

Assuming independence between the weight vectors may seem like a strong assumption. In practice, however, the dependency between the weight vectors is not strong. Intuitively, once the low level layers are fixed, it is reasonable to expect that a good weight vector for a given class will be aligned with the average hidden activations before the softmax layer for that class. That is, units with high average activations should be used for prediction. We further justify the validity of this approximation and present empirical evidence for it in Appendix C.1. There we show that when the softmax weights are retrained multiple times when one class is fixed and the other classes are randomly chosen, the weight vectors obtained

for the fixed class are very similar. This confirms that the activations for a class are more important than which other classes are present during the optimisation.

7.2.2 *Gaussian model*

Possibly the simplest approach consists of modelling $p(\mathrm{W} \mid \widetilde{\mathrm{W}})$ as a Gaussian distribution:

$$p(\mathrm{W} \mid \widetilde{\mathrm{W}}) = \int \mathcal{N}(\mathrm{W} \mid \mu, \Sigma) p(\mu, \Sigma \mid \widetilde{\mathrm{W}}) \, \mathrm{d}\mu \, \mathrm{d}\Sigma. \tag{7.13}$$

Details for this section can be found in Murphy, 2012. The normal-inverse-Wishart distribution for μ and Σ is a conjugate prior for the Gaussian, which allows for the posterior to be written in closed form. More precisely,

$$\begin{aligned} p(\mu, \Sigma) &= \mathcal{NIW}(\mu, \Sigma \mid \mu_0, \kappa_0, \Lambda_0, \nu_0) \\ &= \frac{1}{Z} |\Sigma|^{-(\nu_0 + p)/2 + 1} e^{-\frac{1}{2} tr(\Lambda_0 \Sigma^{-1}) - \frac{\kappa_0}{2} (\mu - \mu_0)^t \Sigma^{-1} (\mu - \mu_0)}, \end{aligned} \tag{7.14}$$

where Z is the normalising constant. The posterior $p(\mu, \Sigma \mid \widetilde{\mathrm{W}})$ also follows a normal-inverse-Wishart distribution:

$$p(\mu, \Sigma \mid \widetilde{\mathrm{W}}) = \mathcal{NIW}(\mu, \Sigma \mid \mu_{\widetilde{N}}, \kappa_{\widetilde{N}}, \Lambda_{\widetilde{N}}, \nu_{\widetilde{N}}), \tag{7.15}$$

where

$$\begin{aligned} \mu_{\widetilde{N}} &= \frac{\kappa_0}{\kappa_0 + \widetilde{N}} \mu_0 + \frac{\widetilde{N}}{\kappa_0 + \widetilde{N}} \overline{\widetilde{\mathrm{W}}} \\ \kappa_{\widetilde{N}} &= \kappa_0 + \widetilde{N} \\ \Lambda_{\widetilde{N}} &= \Lambda_0 + S + \frac{\kappa_0 \widetilde{N}}{\kappa_0 + \widetilde{N}} (\overline{\widetilde{\mathrm{W}}} - \mu_0)(\overline{\widetilde{\mathrm{W}}} - \mu_0)^{\intercal} \\ \nu_{\widetilde{N}} &= \nu_0 + \widetilde{N}, \end{aligned}$$

and S is the sample covariance of $\widetilde{\mathrm{W}}$ and $\overline{\widetilde{\mathrm{W}}}$ denotes the mean weight vector.

For this model, we can integrate Equation (7.13) in closed form, which results in the following multivariate Student t-distribution:

$$p(\mathrm{W} \mid \widetilde{\mathrm{W}}) = t_{\nu_{\widetilde{N}} - p + 1} \left(\mu_{\widetilde{N}}, \frac{\Lambda_{\widetilde{N}} (\kappa_{\widetilde{N}} + 1)}{\kappa_{\widetilde{N}} (\nu_{\widetilde{N}} - p + 1)} \right). \tag{7.16}$$

Alternatively, we can also compute the MAP solutions for the parameters $\theta^{\text{MAP}} = \{\mu^{\text{MAP}}, \Sigma^{\text{MAP}}\}$ and the approximations discussed in Section 7.1 lead to $p(\mathrm{W} \mid \widetilde{\mathcal{D}}) \approx p(\mathrm{W} \mid \widetilde{\mathrm{W}}^{\text{MAP}}) = \mathcal{N}(\mathrm{W} \mid \mu^{\text{MAP}}, \Sigma^{\text{MAP}})$, and the posterior becomes

$$p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathcal{D}}) \propto \mathcal{N}(\mathrm{W} \mid \mu^{\text{MAP}}, \Sigma^{\text{MAP}}) \prod_{n=1}^{N} p(y_n \mid \Phi_\varphi(\mathbf{x}_n), \mathrm{W}). \tag{7.17}$$

For both the analytic posterior and the MAP approximation, $p(\mathrm{W} \mid \widetilde{\mathrm{W}})$ depends on the hyperparameters of the normal-inverse-Wishart distribution: μ_0, ν_0, κ_0 and Λ_0. In practise, it is common to choose uninformative or weakly data dependent priors as discussed by Murphy (2012, Chapter 4). One way would be by optimising the log probability of held-out training weights, see Section 7.3.2 for a discussion.

For few-shot testing we use the MAP estimates for the weights of the new classes. We found that restricting the covariance matrix to be isotropic, $\Sigma = \sigma^2 \mathrm{I}$, empirically performed best, probably due to the small number of effective data points to learn from as mentioned above.

We note that, for the presented Gaussian model, the transfer is limited to a small number of parameters but already has a beneficial effect. The framework we present is general and also allows for more elaborate probabilistic models, which lead to more ambitious concept transfer, such as the Gaussian latent feature model proposed in Griffiths and Ghahramani (2011, Section 5.1).

7.2.3 *Relation of the Gaussian model to logistic regression.*

Standard logistic regression corresponds to the maximum likelihood (MLE) solution of the softmax likelihood $p(y_n \mid \Phi_\varphi(\mathbf{x}_n), \mathrm{W}) = \mathrm{softmax}(\mathrm{W}\Phi_\varphi(\mathbf{x}_n))$. Often, L_2-regularisation on the weights W with inverse regularisation strength $1/C_{\text{reg}}$ is used; the solution to this regularised optimisation problem corresponds to the MAP solution of a model with isotropic Gaussian prior on the weights with zero mean: $p(\mathrm{W} \mid \mathcal{D}) \propto \mathcal{N}(\mathrm{W} \mid 0, \frac{1}{2} C_{\text{reg}} \mathrm{I}) \prod_{n=1}^{N} p(y_n \mid \Phi_\varphi(\mathbf{x}_n), \mathrm{W})$. This method is analogous to Equation (7.17). However, the probabilistic framework has several advantages: (i) modelling assumptions and approximations are made explicit, (ii) it is strictly more general and can incorporate non-zero means μ^{MAP}, whereas standard regularised logistic regression assumes zero mean, and (iii) the probabilistic interpretation provides a principled way of choosing the regularisation constant using the trained weights $\widetilde{\mathrm{W}}$: $C_{\text{reg}} = 2\sigma^2_{\widetilde{\mathrm{W}}}$, where $\sigma^2_{\widetilde{\mathrm{W}}}$ is the empirical variance of the weights $\widetilde{\mathrm{W}}^{\text{MAP}}$. In few-shot learning, alternative (frequentist) methods such as cross-validation suffer in the face of the small number of few-shot examples, and are not applicable in 1-shot learning at all.

7.2.4 *Mixture of Gaussians (GMM)*

A Gaussian mixture model can potentially leverage cluster structure in the weights (animal classes might have similar weights, for example). This is related to the tree-based prior proposed in Srivastava and Salakhutdinov (2013). MAP inference is performed because exact inference is intractable. Similarly to the Gaussian case, different structures for the covariance of each cluster were tested. In our experiments, we fit the parameters of the GMM via maximum likelihood using the EM algorithm. GMMs model $p(\mathrm{W} \mid \widetilde{\mathrm{W}})$ as a mixture of Gaussians with S components:

$$p(\mathrm{W} \mid \widetilde{\mathrm{W}}) = \int \left(\sum_{s=1}^{S} \pi_s \mathcal{N}(\mathrm{W} \mid \mu_s, \Sigma_s) \right) p(\mu_1, \ldots, \mu_S, \Sigma_1, \ldots, \Sigma_S \mid \widetilde{\mathrm{W}}) \, \mathrm{d}\mu_1 \ldots \mathrm{d}\mu_S \, \mathrm{d}\Sigma_1 \ldots \mathrm{d}\Sigma_S, \tag{7.18}$$

where $\sum_{s=1}^{S} \pi_s = 1$. In this work, we only compute the MAP mean and covariance for each of the clusters, as opposed to averaging over the parameters of the mixture. The resulting posterior is

$$p(\mathrm{W} \mid \widetilde{\mathrm{W}}) = \sum_{s=1}^{S} \pi_s \mathcal{N}(\mathrm{W} \mid \mu_{\mathrm{MAP},s}, \Sigma_{\mathrm{MAP},s}). \tag{7.19}$$

The components of the mixture are fit in two ways. For CIFAR-100, the classes are grouped into 20 superclasses, each containing 5 of the 100 classes. One option is therefore to initialise 20 components, each fit with the data points in the corresponding superclass. For each such individual Gaussian, the MAP inference method presented in the previous section can be used. In order to increase the number of weight examples in each superclass, we merge the original superclasses into 9 larger superclasses as follows:

- Aquatic mammals + fish
- flowers + fruit and vegetables + trees
- insects + non-insect invertebrates + reptiles
- medium-sized mammals + small mammals
- large carnivores + large omnivores and herbivores
- people
- large man-made outdoor things + large natural outdoor things

- food containers + household electrical devices + household furniture
- Vehicles 1 + Vehicles 2.

As a second option, the parameters of the mixture can also be fit using maximum likelihood with EM. We use the implementation of EM in scikit-learn. Both 3 and 10 clusters are considered in CIFAR-100. Weight log-likelihoods under this model and few-shot performance are discussed in Section 7.3.2.

Note that, similarly to the Gaussian model, we consider isotropic, diagonal or full covariance models for the covariance matrices.

7.2.5 *Laplace distribution*

Sparsity is an attractive feature which could be helpful for modelling the weights. Indeed, it is reasonable to assume that each class uses a set of characteristic features which drive classification accuracy, while others are irrelevant. Sparse models would then provide sensible regularisation. As such, we consider a product of independent Laplace distributions. Sections 7.2.2 and 7.2.3 highlight the relation between a Gaussian prior on the weights and L_2 regularised logistic regression. One can similarly show that the Laplace prior is related to L_1 regularised logistic regression, which is well known for encouraging sparsity when considering MAP solutions.

We consider a prior which factors along the feature dimensions:

$$p(\widetilde{\mathrm{W}} \mid \{\mu_j\}, \{\lambda_j\}) = \prod_j^p \frac{1}{2\lambda_j} \exp\left(- \sum_i^{\widetilde{C}} \frac{|\widetilde{\mathrm{W}}_{ij} - \mu_j|}{\lambda_j} \right). \tag{7.20}$$

where the product over j is along the feature dimensions and the sum over i is across the classes. We fit the parameters μ and λ via maximum likelihood:

$$\begin{aligned} \mu_{\mathrm{MLE},j} &= \mathrm{median}_i(\widetilde{\mathrm{W}}_{ij}) \\ \lambda_{\mathrm{MLE},j} &= \frac{1}{N} \sum_i |\widetilde{\mathrm{W}}_{ij} - \mu_j|, \end{aligned} \tag{7.21}$$

such that

$$p(\mathrm{W} \mid \widetilde{\mathrm{W}}) = \prod_j^p \frac{1}{2\lambda_{\mathrm{MLE},j}} \exp\left(- \sum_i^C \frac{|\mathrm{W}_{ij} - \mu_{\mathrm{MLE},j}|}{\lambda_{\mathrm{MLE},j}} \right). \tag{7.22}$$

An isotropic Laplace model with mean μ and scale λ is also considered:

$$p(\widetilde{\mathrm{W}} \mid \mu, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{\sum_{ij} |\widetilde{\mathrm{W}}_{ij} - \mu|}{\lambda}\right), \tag{7.23}$$

where

$$\begin{aligned} \mu_{\mathrm{MLE}} &= \mathrm{median}(\widetilde{\mathrm{W}}) \\ \lambda_{\mathrm{MLE}} &= \frac{1}{Np} \sum_{ij} |\widetilde{\mathrm{W}}_{ij} - \mu|, \end{aligned} \tag{7.24}$$

7.2.6 *Approximate inference methods*

In this section we briefly discuss different inference methods for the probabilistic models. For our main comparison to other approaches in Section 7.3 we only considered MAP inference as we found that other more complicated inference schemes do not yield a practical benefit. However, in Section 7.3.2 we provide a detailed model comparison, in which we also consider other approximate inference methods.

In all cases the gradients of the densities with respect to W can be computed, enabling MAP inference in the few-shot learning phase to be efficiently performed via gradient-based optimisation using L-BFGS (Liu and Nocedal, 1989). Alternatively, Markov Chain Monte Carlo (MCMC) sampling can be performed to approximate the associated integral, see Equation (7.1). Due to the high dimensionality of the space and as gradients are available, we employ Hybrid Monte Carlo (HMC) (Neal, 2011) sampling in the form of the recently proposed NUTS sampler that automatically tunes the HMC parameters (step size and number of leapfrog steps) (Hoffman and Gelman, 2014). For the GMMs we employed `pymc3` (Salvatier et al., 2016) to perform MAP inference. Experiments with HMC showed the improvements to be marginal at best such that they do not justify the additional computational cost, see Section 7.3.2 for details.

7.2.7 *Calibration*

While generalisation accuracy is often the key objective when training a classifier, calibration is also a fundamental concern in many applications such as decision making for autonomous driving and medicine. Here, calibration refers to the agreement between a classifier's uncertainty and the frequency of its mistakes. For example, if a classifier predicts $p(y = c \mid \mathbf{x}) = 0.5$ for a particular class c, it is said to be calibrated well if it is correct 50% of the time. Calibration has recently

received more attention, for example, Guo et al. (2017) show that the calibration of deep architectures deteriorates as depth and complexity increase. Here, we follow their description.

To evaluate calibration, consider a classifier over C distinct classes and denote by $\widehat{y}(\mathbf{x})$ its class prediction and by $\widehat{p}(\mathbf{x})$ its confidence as assigned by the classifier. The classifier is said to be perfectly calibrated if:

$$\Pr(\widehat{y}(\mathbf{x}) = y \mid \widehat{p}(\mathbf{x}) = p) = p \qquad \forall\ p \in [0, 1]\,. \tag{7.25}$$

A calibration curve visualises the proportion of examples correctly classified as a function of their predicted probability; a perfectly calibrated classifier should result in a diagonal line. Following Guo et al. (2017), we consider the Expected Calibration Error (ECE) (Naeini et al., 2015) as scalar summary measures of calibration. ECE can be interpreted as the weighted average of the distance of the calibration curve to the diagonal. More formally, we can evaluate this gap between confidence and accuracy on the population as

$$\int_{p\in[0,1]} \mathbb{E}_{\mathbf{x},y}[|\Pr(\widehat{y}(\mathbf{x}) = y \mid \widehat{p}(\mathbf{x}) = p) - p|]\,, \tag{7.26}$$

which is zero for a perfectly calibrated classifier. To estimate ECE from a dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, Guo et al. (2017) propose a histogram approach: First, split the range $[0, 1]$ into M equally sized bins and denote by $\mathcal{B}_m$ the set of indices of samples whose predicted confidence $\widehat{p}(\mathbf{x})$ falls into bin m. Then the accuracy and confidence of $\mathcal{B}_m$ are given as:

$$\text{acc}(\mathcal{B}_m) = \frac{1}{|\mathcal{B}_m|} \sum_{n\in\mathcal{B}_m} \mathbb{1}(\widehat{y}(\mathbf{x}_n) = y_n) \tag{7.27}$$

$$\text{conf}(\mathcal{B}_m) = \frac{1}{|\mathcal{B}_m|} \sum_{n\in\mathcal{B}_m} \widehat{p}(\mathbf{x}_n)\,. \tag{7.28}$$

The calibration error ECE can then be estimated as (Guo et al., 2017, Equation (3)):

$$\text{ECE} = \sum_{m=1}^{M} \frac{|\mathcal{B}_m|}{N} |\text{acc}(\mathcal{B}_m) - \text{conf}(\mathcal{B}_m)|\,. \tag{7.29}$$

7.3 EMPIRICAL EVALUATION

7.3.1 *Datasets*

We use two benchmark datasets to compare our model to alternative approaches, CIFAR-100 and *mini*ImageNet. CIFAR-100 (Krizhevsky, 2009) consists of 100

classes each with 500 training and 100 test images of size 32×32. The classes are grouped into 20 superclasses with 5 classes each. For example, the superclass "fish" contains the classes aquarium fish, flatfish, ray, shark, and trout[1]. Unless otherwise stated, we used a random split into 80 meta-train classes and 20 meta-test classes.

There exists a range of few-shot learning problems that have been derived from the ImageNet dataset (Russakovsky et al., 2015), most notably *mini*ImageNet (Vinyals et al., 2016), which has become a standard testbed for few-shot learning. It is derived from the ImageNet ILSVRC12 dataset (Russakovsky et al., 2015) by extracting 100 out of the 1000 classes. Each class contains 600 images downscaled to 84×84 pixels. Ravi and Larochelle (2017) proposed to split these 100 classes into 64 (meta-)train, 16 validation, and 20 (meta-)test classes. As our approach does not require a validation set, we used the 16 validation classes for representational learning as well.

7.3.2 *Model comparison on CIFAR-100*

We performed an extensive comparison between different probabilistic models of the weights using several inference procedures as detailed in Section 7.2. In this section we report results on the CIFAR-100 dataset for (i) Gaussian, (ii) mixture of Gaussians, and (iii) Laplace models, all with either MAP estimation or Hybrid Monte Carlo sampling.

Summary

We found that the simple Gaussian model is on par with or outperforms other methods at few-shot learning, which we attribute to it striking a good balance between choosing a complex model, which may better fit the weights, and statistical efficiency, as the number of weights $\widetilde{C}$ (80 in our case) is often smaller than the dimensionality of the feature representation (256 in our case). This finding is supported by computing the log-likelihood of held-out training weights under such models, with the Gaussian model performing best. Experiments using Hybrid Monte Carlo sampling for few-shot learning returned very similar performance to MAP estimation and at a much higher computational cost, due to the difficulty of performing sampling in such a high dimensional parameter space. Our recommendation is that when the number of initial classes is small relative to the hidden layer size, practitioners should use simple models and employ simple inference schemes to estimate all free parameters, thereby avoiding expending valuable data on validation sets.

1 Swallows, both European and African, are notably absent in CIFAR-100

Experiments

In the following we report the detailed model comparison on CIFAR-100, both for the model of the weights $p(\mathrm{W} \mid \widetilde{\mathrm{W}})$ and for the inference procedure at meta-test time (MAP or Hybrid Monte Carlo (HMC) sampling using NUTS (Hoffman and Gelman, 2014). We report the log-likelihood of the weights under different models, as well as the accuracy, log-likelihood and calibration in the few-shot learning task. Table 7.1 and Table 7.2 show descriptions of the methods analysed for phase 2 (concept learning and transfer) and phase 3 (few-shot learning) of our few-shot learning pipeline described in Section 7.1.

Method name	**Phase 2: Concept learning**	
	Prior distribution	Inference
Gauss (iso)	Gaussian isotropic covariance	MAP
Gauss (MAP prior)	Gaussian isotropic covariance	MAP
Gauss (integr. prior)	Gaussian full covariance	Integrated
GMM (supercl.)	GMM on superclasses iso. cov.	MAP
GMM (3, iso)	GMM on 3 clusters iso. cov.	MLE
GMM (3, diag)	GMM on 3 clusters diagonal cov.	MLE
GMM (10, iso)	GMM on 10 clusters iso. cov.	MLE
Laplace (diag)	Laplace diagonal covariance	MLE

Table 7.1: Description of the inference for the parameters of the prior in phase 2 (concept learning) for the models analysed in Figure 7.4. This specifies the inference procedure for θ in $p(\mathbf{w} \mid \theta)$ after observing the training weights $\widetilde{\mathrm{W}}$.

Method name	**Phase 3: Few-shot learning**	
	Prior distribution	Inference
Gauss (iso) MAP	Gaussian	MAP
Gauss (MAP prior) MAP	Gaussian	MAP
Gauss (MAP prior) HMC	Gaussian	HMC
Gauss (integr. prior) MAP	Gaussian	MAP
Gauss (integr. prior) HMC	Gaussian	HMC
GMM (supercl.) MAP	GMM on superclasses	MAP
GMM (3, iso) MAP	GMM on 3 isotropic comp.	MAP
Laplace (diag) HMC	Laplace (diagonal)	HMC
Laplace (diag) MAP	Laplace (diagonal)	MAP

Table 7.2: Methods and inference procedure during phase 3 (few-shot learning) for the models used in Figure 7.4. This specifies the inference procedure used when computing $p(\mathrm{W} \mid \mathcal{D}, \widetilde{\mathrm{W}})$ for the specified prior distribution, see Equation (7.8) (MAP) and Equation (7.9) (HMC)

Here, we report results on a VGG-like architecture. Preliminary results when switching from VGG to a ResNet architecture lead to the same conclusion as

experiments on *mini*ImageNet in Section 7.3.7: deeper features consistently lead to higher few-shot performance on all methods whereas the ordering of the methods stays roughly the same.

Analysis of the models on held-out training weights.

Model	Optimised value of mean log probability
Gauss (iso)	175.9 ± 0.3
Gauss (MAP prior)	196.1 ± 0.5
Gauss (integr. prior)	200.6 ± 0.4
GMM 3-means (iso)	179.0 ± 0.3
GMM 3-means (diag)	181.2 ± 0.3
GMM 10-means iso	181.6 ± 0.4
GMM 10-means (diag)	181.6 ± 0.4
Laplace (iso)	173.8 ± 0.4
Laplace (diag)	176.6 ± 0.5

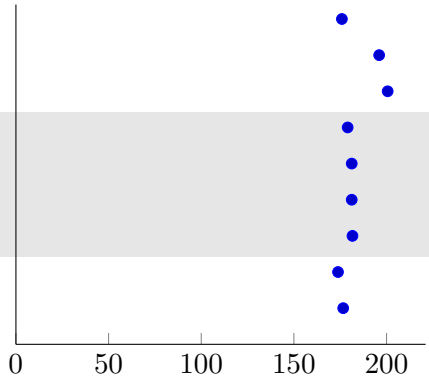


Table 7.3: Held-out log probabilities on random 70/10-splits of the training weights for the different models on CIFAR-100 (higher is better). Values are averaged over 50 splits.

First, we analyse how well the different priors for the new softmax weights are able to fit the $\widetilde{C} = 80$ training weights $\widetilde{\mathrm{W}}$. We randomly excluded 10 of these weights and evaluated their held-out log likelihood given the remaining $\widetilde{C} - 10 = 70$ weights. We emphasise that this approach also constitutes a principled way to set the hyperparameters of the prior and, critically, relies on an explicit probabilistic model.

The log probabilities are averaged over 50 random splits and results with the best optimised hyperparameter values are shown in Table 7.3 for CIFAR-100 (higher is better). We find that all models behave very similarly but multivariate Gaussian models generally outperform other models. We attribute the good performance of simpler models to the small number of data points ($\widetilde{C} - 10 = 70$ training weights) and the high dimensionality of the space, which means that fitting even simple models is difficult.

Few-shot performance in CIFAR-100.

Accuracies are measured on a 5-way classification task on the new meta-test classes for k-shots with $k \in \{1, 5, 10\}$. Results were averaged two-fold: (i) 20 random splits of the 5 classes; (ii) 10 repetitions of each split with different few-shot training examples. Among our models, no statistically significant difference in accuracy is observed, with the exception of Laplace MAP and GMM (iso), which consistently

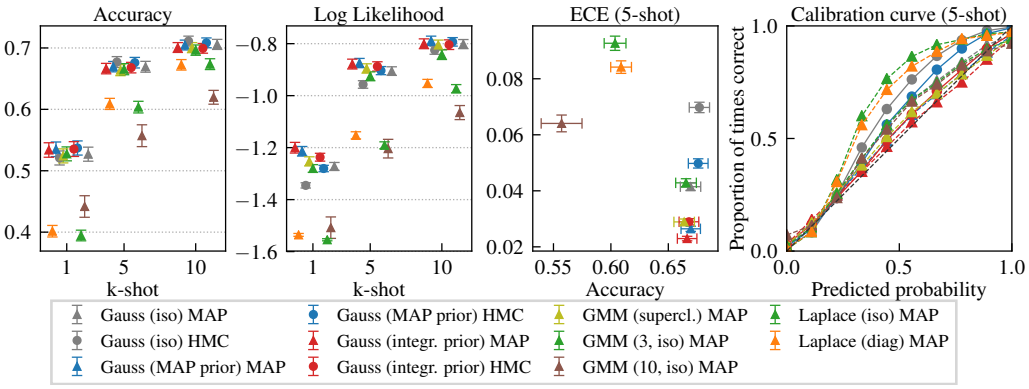


Figure 7.4: Results on CIFAR-100 for a CNN with VGG style architecture. We report accuracy, log-likelihood and calibration for the methods and inference procedures presented in Table 7.2. With the exception of GMM (10, iso) and Laplace, all methods behave similarly in terms of accuracy and log-likelihood. Gauss (integr. prior) HMC and Gauss (MAP) HMC are slightly better calibrated than our proposed Gauss (MAP) iso, but require significantly more computation for the sampling procedure.

underperform. These findings are consistent in terms of log-likelihoods, see the first and second plots in Figure 7.4.

Our methods are well calibrated, with Gaussian models generally better than Laplace models. Moreover, all methods (with the exception of Laplace and GMM (10, iso) have low ECE and high accuracy, see the third and fourth plots of Figure 7.4. While Gauss (integr. prior) HMC and Gauss (MAP) HMC are slightly better calibrated than our proposed method, Gauss (MAP) iso, we believe the gain in calibration is not worth the significant increase in computational resources needed for the sampling procedure. Interestingly, both GMM approaches are not able to outperform the other, simpler models. This is in line with the previous observation that the simpler models are better able to explain the weights. Again, we attribute this inability of mixture models to use their larger expressiveness and capacity to the small number of data points and the high-dimensionality of weight-space, which means fitting even simple models is difficult. These observations suggest that the use of mixture models in this type of few-shot learning framework is not beneficial and is in contrast to the approach of Srivastava and Salakhutdinov (2013), who employ a tree-structured mixture model. The authors compare a model in which the assignments to the superclasses in the tree are optimised against a model with a naive initialisation of the superclass assignments, and show that the former outperforms the latter. However, they do not compare against a simpler baseline, such as our proposed single Gaussian model.

Overall, we observe that there is no significant benefit to more complex methods over the simple isotropic Gaussian, either in terms of accuracy, log-likelihood, or calibration. Thus, our recommendation is that practitioners should use simple

models and employ simple inference schemes to estimate all free parameters, which avoids expending valuable data on validation sets.

7.3.3 *Representational learning*

We employ standard CNNs inspired by ResNet-34 (He et al., 2016) and VGG (Simonyan and Zisserman, 2015) for the representational learning on the $\widetilde{C}$ base classes (phase 1 in Section 7.1). These trained networks provide both $\widetilde{\mathrm{W}}^{\mathrm{MAP}}$ and the fixed feature representation Φ_φ for the few-shot learning and testing. We employed standard data augmentation from ImageNet for the representational learning but highlight that no data augmentation was used during the few-shot training and testing to consistently compare with other approaches.

An important design decision in our approach is the output dimensionality d_Φ of the feature extractor, to which the heads are attached. Standard ResNet and VGG architectures use up to $d_\Phi = 4096$ dimensions, which would further aggravate the "large d, small N" problem of the classifier weights. There is a certain arbitrariness connected with the exact choice of architecture; however, initial experiments suggested that both too small ($d_\Phi < 64$) as well as too large ($d_\Phi > 512$) values decreased performance, with $d_\Phi \in \{128, 256\}$ providing the best trade-off between expressiveness and complexity, such that we chose these in practice. For further details on the architecture, training, and data augmentation see Appendix B.1.2.

t-SNE embeddings (Maaten and Hinton, 2008) of the learned last layer weights show sensible clusters, which highlights the structure that can be exploited by the probabilistic model (see Figure 7.1).

7.3.4 *Baselines and competing methods*

We compare against several baselines as well as recent state-of-the-art methods mentioned in Section 6.4. The baselines are computed on the features $\Phi_\varphi(\mathbf{x})$ from the last hidden layer of the trained CNN: (i) Nearest Neighbours with cosine distance and (ii) regularised logistic regression ("Log Reg") with regularisation constant set by cross-validation ("Log Reg (cv)"). We also compare against three recent few-shot methods: (i) Matching Networks[2] (Vinyals et al., 2016), (ii) Prototypical Networks, with numbers reported from Snell et al., 2017, (iii) Meta-learner LSTM, with numbers reported from Ravi and Larochelle, 2017, and (iv) Model-agnostic Meta-learning (MAML), with numbers reported from Finn et al., 2017.

2 as reimplemented and optimised by https://github.com/AntreasAntoniou/MatchingNetworks to produce results that are superior to those originally published.

In addition to our proposed Gaussian model, we also consider the case of regularised logistic regression with regularisation constant determined by the variance of the weights, $\sigma_{\widetilde{W}}$, as motivated by our probabilistic framework, refer to the discussion in Section 7.2.3. This case is commonly ignored in other works on few-shot learning and constitutes a proposed method rather than a baseline.

7.3.5 *Testing protocol*

We evaluate the methods on 600 random few-shot tasks by randomly sampling 5 classes from the 20 test classes and perform 5-way few-shot learning. Following Snell et al. (2017), we use 15 randomly selected images per class for few-shot testing to compute accuracies and calibration.

7.3.6 *Overall few-shot performance*

Method	1-shot	5-shot
ResNet-34 + Isotropic Gaussian (ours)	$\mathbf{56.3 \pm 0.4\%}$	$\mathbf{73.9 \pm 0.3\%}$
Matching Nets (1-shot)[1]	$46.8 \pm 0.5\%$	-
Matching Nets (5-shot)[1]	-	$62.7 \pm 0.5\%$
Meta-Learner LSTM (Ravi and Larochelle, 2017)	$43.4 \pm 0.8\%$	$60.6 \pm 0.7\%$
Prototypical Nets (1-shot) (Snell et al., 2017)	$49.4 \pm 0.8\%$	$65.4 \pm 0.7\%$
Prototypical Nets (5-shot) (Snell et al., 2017)	$45.1 \pm 0.8\%$	$68.2 \pm 0.7\%$
MAML (Finn et al., 2017)	$48.7 \pm 1.8\%$	$63.1 \pm 0.7\%$

[1] reimplemented from https://github.com/AntreasAntoniou/MatchingNetworks

Table 7.4: Accuracy on 5-way classification on *mini*ImageNet. Our best method, an isotropic Gaussian model using ResNet-34 features consistently outperforms all competing methods by a wide margin. 10-shot performance for our method is $78.5 \pm 0.3\%$. Competing methods use a simpler 4-layer convolutional feature extractor backbone.

We report performance on the *mini*ImageNet dataset in Table 7.4 and Figures 7.5 and 7.6. The best method uses as feature extractor a modified ResNet-34 with 256 features, trained with all 600 examples per training class, and a simple isotropic Gaussian model on the weights for concept learning and transfer. Despite its simplicity, our method achieves state-of-the-art and beats prototypical networks by a wide margin of about 6%. The baseline methods using the same feature extractor are also state-of-the-art compared to prototypical networks and both logistic regressions show comparable accuracy to our methods except for on 1-shot learning. In terms of log-likelihoods, Log Reg ($C = 2\sigma^2_{\widetilde{W}}$) fares slightly better, whereas Log Reg (cv) is much worse.

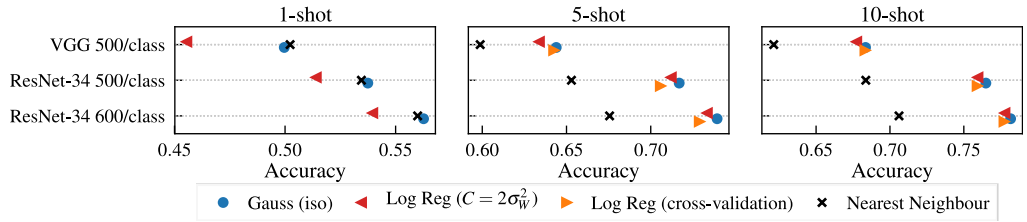


Figure 7.5: Comparison of different network architectures and training set sizes on the few-shot learning task: VGG style network (trained on 500 images per class) and ResNet-34 style network (trained on 500 and 600 images per class, respectively). Both, deeper networks and larger number of training images, give rise to features that transfer better to few-shot learning.

7.3.7 *Deeper features lead to better few-shot learning*

We investigate the influence of different feature extractors of increasing complexity on performance in Figures 7.5 and 7.6: (i) a VGG style network (500 training images per class), (ii) a ResNet-34 (500 examples per class), and (iii) a ResNet-34 (all 600 examples per class). We find that the complexity of the feature extractor as well as training set size consistently correlate with the accuracy at few-shot time. For instance, on 5-shot, Gauss (iso) achieves 65% accuracy with a VGG network and 74% with a ResNet trained with all available training data, a significant increase of almost 10%. Moreover, Gauss (iso) outperforms Log Reg ($C = 2\sigma^2_{\widehat{W}}$) on 1-shot learning for all feature extractors, and performs similarly on 5- and 10-shot. We attribute the difference to the former's ability to also model the mean of the Gaussian, whereas logistic regression assumes a zero mean.

Importantly, this result implies that training specifically for a particular way-shot-combination is not necessary for achieving high generalisation performance on this few-shot problem. On the contrary, training a powerful deep feature extractor on batch classification using all available training data, then building a simple probabilistic model using the learned features and weights, achieves state-of-the-art. Recent models that use episodic training cannot leverage such deep feature extractors, as for them the depth of the model is limited by the nature of training itself, see also our discussion in Section 8.5.

We ran additional experiments with Matching Networks on *mini*ImageNet in which we added one or two additional convolutional layers to the architecture from the original paper. The resulting test accuracy remained unchanged.

The reference baseline in the few-shot learning literature is nearest neighbours, which performs on par with Gauss (iso) on 1-shot learning but is outperformed by all methods on 5- and 10-shot. This is evidence that building a simple classifier on top of the learned features works significantly better for few-shot learning than nearest neighbours.

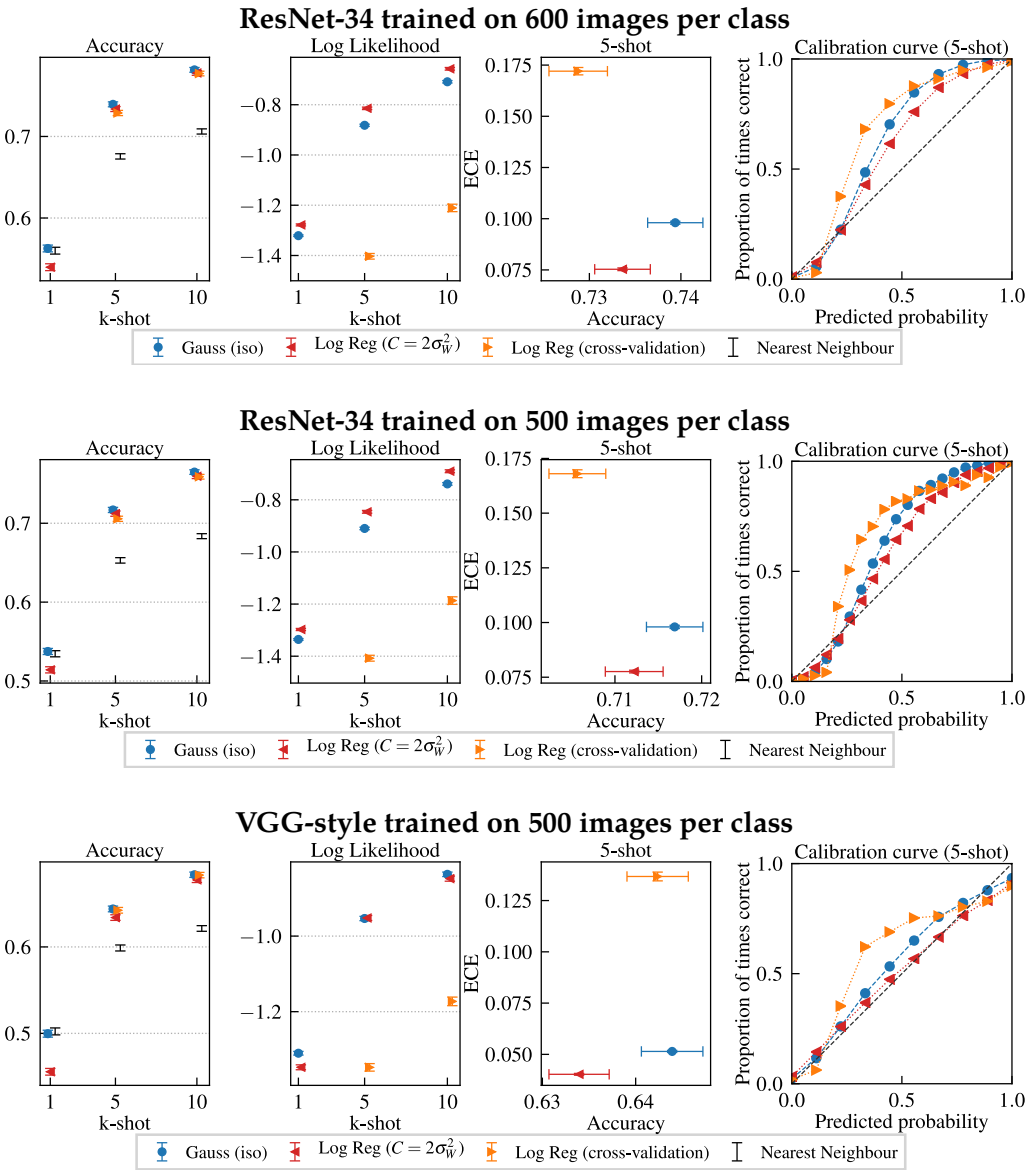


Figure 7.6: Probabilistic few-shot learning results for the *mini*ImageNet dataset utilising different network architectures and representational training. *top:* a ResNet-34 trained with all 600 examples per class; *middle:* a ResNet-34 trained with 500 images per class; *bottom:* a VGG style network trained with 500 images per class. We highlight that for all three architectures the order of the different methods as well as the main messages are the same. However, the general performance in terms of accuracy and calibration differ between the architectures. The more complex architecture trained on most images performs best in terms of accuracy, indicating that it learns better features for few-shot learning. Both ResNets behave very similarly on calibration whereas the VGG-style network performs better (lower ECE and higher log likelihood as well as more diagonal calibration curve). This is in line with observations by Guo et al., 2017 that calibration of deep architectures gets worse as depth and complexity increase.

7.3.8 *Calibration*

We find that Log Reg ($C = 2\sigma^2_{\widetilde{W}}$) and Gauss (iso) provide better accuracy and calibration than Log Reg (cross-validation), see Figure 7.6. Moreover, we find that the deeper and more complex ResNet feature extractors give rise to less well calibrated models compared to a shallower and simpler VGG network. This is in line with observations by Guo et al. (2017) that calibration of deep architectures deteriorates with depth and complexity. The difference in calibration quality for different regularisations of logistic regression highlights the importance of choosing the right constant, as we discuss now.

7.3.9 *Choice of the regularisation constant for logistic regression*

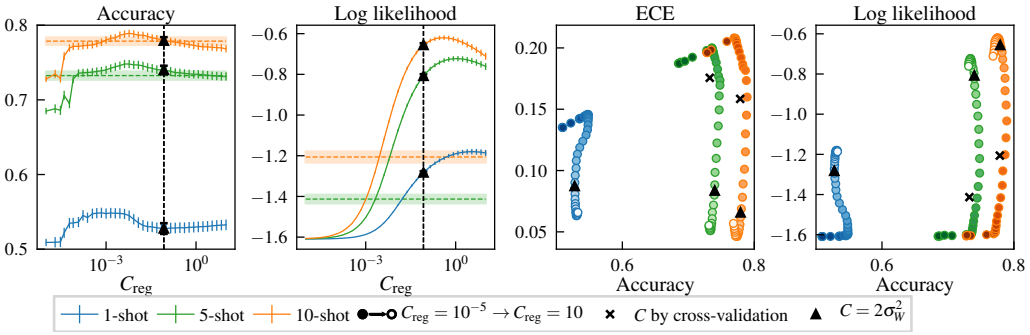


Figure 7.7: Choice of regularisation constant for logistic regression for few-shot learning. Summary of accuracy and calibration in terms of log likelihood and Expected Calibration Error (ECE). Results for $C_{\text{reg}} = 2\sigma^2_{\widetilde{W}}$ are drawn as black triangles. Dashed lines correspond to logistic regression with cross-validated regularisation constant. Colour brightness of the markers for the right most plots ranges from dark ($C = 10^{-5}$) to bright ($C = 10$). Calibration in terms of ECE (lower is better) is consistent with log likelihoods (higher is better): The Bayesian inspired choice of the regularisation parameter strikes a good balance between accuracy and calibration and consistently outperforms the cross-validated choice of the parameter.

The results so far suggest that training a simple linear model such as regularised logistic regression might be sufficient to perform well in few-shot learning. However, while the accuracy at few-shot time does not vary dramatically as the regularisation constant changes, the calibration does, and jointly maximising both quantities is not possible, see the first two plots of Figure 7.7. The standard (frequentist) method to tune this constant is cross-validation, which is not applicable in the 1-shot setting, and suffers from lack of data in 5- and 10-shot. Contrariwise, our probabilistic framework provides a principled way of selecting this regularisation parameter by transfer from the training weights: Log Reg ($C = 2\sigma^2_{\widetilde{W}}$) strikes a good balance between accuracy and log-likelihood.

The rightmost plots in Figure 7.7 report ECE and log-likelihood as a function of accuracy and provide further visualisation of the achieved trade-off between accuracy and calibration for Log Reg ($C = 2\sigma^2_{\widetilde{\mathrm{W}}}$), as well as the failure of Log Reg (cross-validation) to achieve a good compromise in 5- and 10-shot.

7.3.10 *Evaluation in an online setting*

We also briefly consider the online learning setting, for which catastrophic forgetting is a well known problem (French, 1999): We jointly test on the 80 old and the 5 new classes. A Bayesian framework naturally extends to this online case, as a (Bayesian) posterior distribution can always be treated as a new prior to incorporate newly arriving data. During few-shot learning and testing we employ a softmax likelihood, which includes both the new and the old weights, resulting in a total of 85 weight vectors; note that we only update the weights for the new classes. We utilise the ResNet-34 trained on 500 images per class to retain 100 test images on the old classes. While the few-shot weights were modelled probabilistically, we use the MAP estimate $\widetilde{\mathrm{W}}^{\mathrm{MAP}}$ for the old weights obtained through standard deep learning. Accuracies are reported in Figure 7.8 for (i) all the 85 classes, (ii) the old 80 classes only, and (iii) the new 5 classes only. For 5- and 10-shot, Gauss (iso) and Log Reg ($2\sigma^2_{\widetilde{W}}$) only lose a couple of percentage points on the accuracy of the old classes, and perform well on the new classes, striking a good trade-off between forgetting and learning at few-shot time. For unregularised (MLE) logistic regression, the new weights completely dominate the old ones, highlighting that the right regularisation is important. Yet, cross-validation in this setting is often very challenging. When training logistic regression without including the old weights ("only new"), the new weights are dominated by the old ones and fail to learn the new classes, making training in the presence of the old weights an essential component for online learning. Our probabilistic framework automatically enforces treatment of the old weights in the likelihood as we also model them in the prior in this case.

7.4 SUMMARY

In this chapter we addressed the few-shot learning task from a transfer learning perspective. Our probabilistic framework for few-shot learning exploits the powerful features and class information learned by a neural network on a large repository dataset. A probabilistic model on the weights of the top level classifier was then used to transfer statistical structure to new classes. Trained on the large repository of related classes, these weights can effectively be treated as observations and used to make inferences on weight vectors of new classes. We

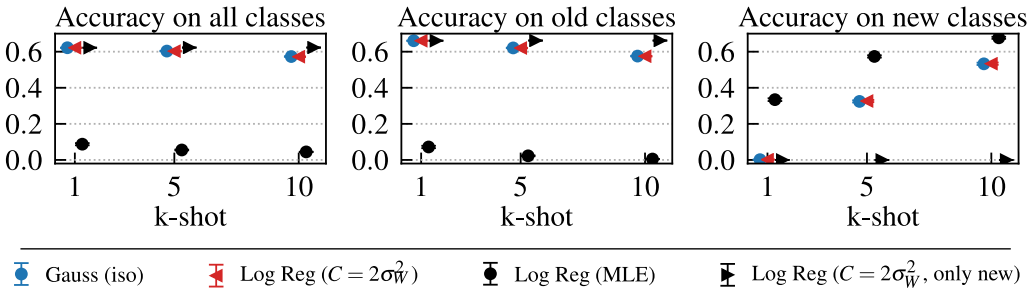


Figure 7.8: Online learning with ResNet-34 features. Gauss (iso) and Log Reg ($2\sigma^2_{\widetilde{W}}$) strike a good trade-off between learning on new classes and forgetting of old classes. Unregularised Log Reg (MLE) and Log Reg ($2\sigma^2_{\widetilde{W}}$, only new), which has not been trained in the presence of the old weights, either completely forget the old classes or do not learn anything, respectively.

then combined this new posterior-turned-prior with likelihoods from the few actual observations for these classes. The resulting second posterior on the new weights can then be used to make probabilistic predictions at test time. While new classifier weights are inferred for this task, we reuse the learned deep features from the old classes without adaptation to the new classes.

The probabilistic model for concept transfer is key and the final performance is influenced by inductive biases in the model as well as the choice of the prior. First, we made a context independence assumption for the weight vectors, which allowed us to model new weights independently of each other. Then, due to the small number of weight vector observations on the repository, we restricted ourselves to very simple models and investigated Gaussian, Laplace and mixture of Gaussians models. We found that Gaussian priors provided the the best trade-off and interpreted our model as performing softmax regression using deep features but with automatically tuned regularisation parameters. This is particularly important in few-shot learning, as it is a low-data regime, in which cross-validation performs poorly and where it is important to train on all available data, rather than using validation sets.

Experiments on *mini*ImageNet using a simple Gaussian model within our framework achieved state-of-the-art results for 1-shot and 5-shot learning by a wide margin, and at the same time returned well calibrated predictions. This finding is contrary to the belief that episodic training is necessary to learn good few-shot features and puts the success of recent complex meta-learning approaches to few-shot learning into context. Our approach is flexible and extensible, being applicable to general discriminative models and few-shot learning paradigms. For example, preliminary results on online few-shot learning indicated that the probabilistic framework mitigates catastrophic forgetting by automatically balancing performance on the new and old classes.

8

META-LEARNING PROBABILISTIC INFERENCE FOR PREDICTION

In this chapter, we consider an extension of the probabilistic few-shot learning approach from Chapter 7. Similarly, we start from a multi-task graphical model with a global feature extractor and an adaptable head model. However, here we embrace the meta-learning perspective (see Section 6.1) and propose an amortised head model that can adapt to new tasks more quickly and flexibly. Our new system is trained episodically, that is, we sub-sample many small tasks from the large repository of images during the meta-training phase, see Figure 6.2.

This proposed few-shot classification method, which we refer to as VERSA, is an example of a more general framework for data efficient and versatile meta-learning that we refer to as *Meta-Learning approximate Probabilistic Inference for Prediction* (ML-PIP). The framework incorporates three key elements. First, we leverage shared statistical structure between tasks via hierarchical probabilistic models developed for multi-task and transfer learning (Heskes, 2000; Bakker and Heskes, 2003). Second, we share information between tasks about how to learn and perform inference using meta-learning (Naik and Mammone, 1992; Thrun and Pratt, 2012; Schmidhuber, 1987). Since uncertainty is rife in small datasets, we provide a procedure for meta-learning probabilistic inference. Third, we enable fast learning that can flexibly handle a wide range of tasks and learning settings via amortisation (P. and Welling, 2014; Rezende et al., 2014). ML-PIP extends existing probabilistic interpretations of meta-learning (Grant et al., 2018) to cover a broad class of methods as we discuss in Section 8.3.

Building on this framework, VERSA substitutes optimisation procedures at test time with forward passes through inference networks. This amortises the cost of inference, resulting in faster test-time performance, and relieves the need to compute second order derivatives during training. VERSA employs a flexible amortisation network that takes few-shot learning datasets as input, and outputs a distribution over task-specific parameters in a single forward pass. The network can handle arbitrary numbers of shots, and for classification, arbitrary numbers of classes at train and test time (see Section 8.2). In Section 8.4, we evaluate VERSA on (i) standard benchmarks where the method sets new state-of-the-art results, (ii) settings where test conditions (shot and way) differ from training, and (iii) a challenging one-shot view reconstruction task.

Interestingly, VERSA does not rely on an explicit prior for the adaptable task specific parameters but learns the structure implicitly.

The work presented in this chapter is joint work with Johnathan Gordon and John Bronskill as well as Sebastian Nowozin and Richard Turner. It is based on the conference paper 'Meta-Learning Probabilistic Inference for Prediction' (Gordon*, Bronskill* et al., 2019) as well as the two earlier workshop papers 'Consolidating the Meta-Learning Zoo: A Unifying Perspective as Posterior Predictive Inference' (Gordon*, Bronskill*, Bauer* et al., 2018a) and 'Versa: Versatile and Efficient Few-shot Learning' (Gordon*, Bronskill*, Bauer* et al., 2018b). My main contributions were to jointly develop the approach and entailed models and jointly devise and partly execute the experiments.

8.1 META-LEARNING PROBABILISTIC INFERENCE FOR PREDICTION

We now present our framework that consists of (i) a multi-task probabilistic model (Section 8.1.1), and (ii) a method for meta-learning probabilistic inference (Section 8.1.2). Thus, in contrast to the previous chapter, we follow the meta-learning perspective introduced in Section 6.1, in which we treat a distribution of tasks t and train our model episodically. Refer back to Figure 6.2 for an illustration of this setting.

8.1.1 *Probabilistic model*

Two principles guide our choice of probabilistic model. First, the use of discriminative models to maximise predictive performance on supervised learning tasks (Ng and Jordan, 2002). Second, the need to leverage shared statistical structure between tasks (i.e. multi-task learning).

Both criteria are met by the standard multi-task directed graphical model shown in Figure 8.1 that employs shared parameters φ, which are common to all tasks, and task specific parameters $\{\psi^{(t)}\}_{t=1}^{T}$. Inputs are denoted by $\mathbf{x}$ and outputs y. We explicitly distinguish between training data $\mathcal{D}^{(t)} = \{(\mathbf{x}_n^{(t)}, y_n^{(t)})\}_{n=1}^{N_t}$, and test data $\{(\mathbf{x}_m^{*(t)}, y_m^{*(t)})\}_{m=1}^{M_t}$ for each task t, as this is how the final meta-test task is structured.

Let $X^{(t)}$ and $\mathbf{y}^{(t)}$ denote all the inputs and outputs (both test and train) for task t. The joint probability of the outputs and task specific parameters for T tasks,

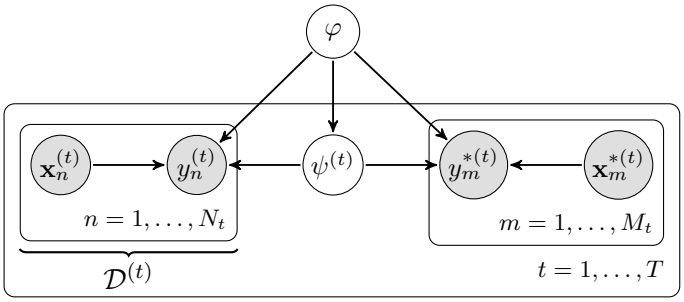


Figure 8.1: Directed graphical model for multi-task learning.

given the inputs and the global parameters φ, is given by (also see the graphical model in Figure 8.1):

$$p\left(\{\mathbf{y}^{(t)}, \psi^{(t)}\}_{t=1}^{T} \mid \{X^{(t)}\}_{t=1}^{T}, \varphi\right) = \prod_{t=1}^{T} p\left(\psi^{(t)} \mid \varphi\right) \prod_{n=1}^{N_t} p\left(y_n^{(t)} \mid \mathbf{x}_n^{(t)}, \psi^{(t)}, \varphi\right) \times \\ \times \prod_{m=1}^{M_t} p\left(y_m^{*(t)} \mid \mathbf{x}_m^{*(t)}, \psi^{(t)}, \varphi\right). \quad (8.1)$$

Our ultimate goal is to perform probabilistic predictions for unseen tasks t. Thus, we are interested in the posterior predictive distribution

$$p(y^{*(t)} \mid \mathbf{x}^{*(t)}, \mathcal{D}^{(t)}, \varphi) = \int p(y^{*(t)} \mid \mathbf{x}^{*(t)}, \psi^{(t)}, \varphi) p(\psi^{(t)} \mid \mathcal{D}^{(t)}, \varphi) \, \mathrm{d}\psi^{(t)}. \quad (8.2)$$

However, this integral cannot be solved in closed form. In the following, we present a framework to meta-learn fast and accurate approximations to the posterior predictive distribution.

8.1.2 *Amortised probabilistic inference*

In this section we present our framework for meta-learning approximate inference that is a simple reframing and extension of existing approaches (Finn et al., 2017; Grant et al., 2018). We again employ point estimates for the shared parameters φ since data across all tasks will pin down their value, similarly to how we treated the feature extractor Φ_φ in Chapter 7. Distributional estimates are used for the task-specific parameters $\psi^{(t)}$ because only a few examples constrain them.

Once the shared parameters φ are learned, the probabilistic solution to fewshot learning in the model above comprises two steps: First, form the posterior distribution over the task-specific parameters $p(\psi^{(t)} \mid \mathbf{x}^{*(t)}, \mathcal{D}^{(t)}, \varphi)$. Second, compute the posterior predictive $p(y^{*(t)} \mid \mathbf{x}^{*(t)}, \varphi)$. Both of these steps require approximations and the emphasis here is on performing this quickly at test time.

We describe the form of the approximation, the optimisation problem used to learn it, and how to implement this efficiently below.

We initially suppress the superscript (t) to denote tasks to reduce notational clutter, but we will reintroduce it at the end of this section when presenting the final training objective (Equation (8.15)).

Specification of the approximate posterior predictive distribution.

Our framework approximates the posterior predictive distribution by an amortised variational distribution $q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi)$ with variational parameters υ. That is, we learn a feed-forward inference network with parameters υ that takes any training dataset $\mathcal{D}$ and test input $\mathbf{x}^*$ as inputs and returns the predictive distribution over the test output y^*. We construct this approximate posterior predictive by first factorising it according to the true posterior predictive in Equation (8.2), and then approximating the posterior of the task dependent parameters $p(\psi \mid \mathcal{D}, \mathbf{x}^*, \varphi)$ by an amortised variational distribution $q_\upsilon(\psi \mid \mathcal{D}, \mathbf{x}^*, \varphi)$:

$$\begin{aligned} q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi) &= \int p(y^* \mid \mathbf{x}^*, \psi, \varphi) q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi) \, \mathrm{d}\psi \\ &= \mathbb{E}_{q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi)} \left[p(y^* \mid \mathbf{x}^*, \psi, \varphi) \right] . \end{aligned} \tag{8.3}$$

Evaluating the involved integral may require additional approximations, for example, Monte Carlo sampling.

In general, the amortisation enables us to perform fast predictions at test time, because the parameters of $q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi)$ are inferred by a forward-pass of a network rather than through test-time optimisation. The form of the variational distribution is identical to those used in other approaches utilising amortised variational inference (Edwards and Storkey, 2017; P. and Welling, 2014): We use a factorised Gaussian distribution for $q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi)$ with means and log-variances determined by the amortisation network. Note that the recognition network q_υ not only amortises over the inputs $\mathbf{x}^*$ but also the training sets $\mathcal{D}$. We explain this further in Section 8.2. While other works employing amortised inference typically use an evidence lower bound (ELBO) to train the amortisation network, we use a different objective that we explain next.

Objective for meta-learning the approximate posterior predictive distribution.

Here we introduce the objective function to train the global and task-specific parameters. We briefly motivate it here and provide a generalised derivation grounded in Bayesian decision theory in Section 8.1.3.

The main quantity of interest in our approach is the posterior predictive distribution $p(y^* \mid \mathbf{x}^*, \mathcal{D})$, which we approximate by a variational distribution $q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi)$. The quality of the approximate posterior predictive for a single

task can be measured by the KL-divergence between the true and approximate posterior predictive distribution

$$\mathrm{KL}(p(y^* \mid \mathbf{x}^*, \mathcal{D}) \parallel q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi)) \,. \tag{8.4}$$

The goal of learning should therefore be to minimise the expected value of this KL averaged over tasks,

$$\upsilon^* = \arg\min_\upsilon \mathbb{E}_{p(\mathcal{D})} \left[\mathrm{KL}(p(y^* \mid \mathbf{x}^*, \mathcal{D}) \parallel q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi))\right] \tag{8.5}$$

$$= \arg\max_\upsilon \mathbb{E}_{p(y^*, \mathcal{D})} \left[\log \int p(y^* \mid \mathbf{x}^*, \psi, \varphi) q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi) \,\mathrm{d}\psi\right] . \tag{8.6}$$

Training will therefore return parameters υ that best approximate the posterior predictive distribution in an average KL sense. So, if the approximate posterior $q_\upsilon(\psi \mid \mathbf{x}^*, \mathcal{D}, \varphi)$ is rich enough, global optimisation will recover the true posterior $p(\psi \mid \mathbf{x}^*, \mathcal{D})$ (assuming $p(\psi \mid \mathbf{x}^*, \mathcal{D})$ obeys identifiability conditions (Casella and Berger, 2002)).[1] Thus, this amortised procedure meta-learns approximate inference that supports accurate predictions.

Equation (8.6) indicates how training could proceed: (i) select a task t at random, (ii) sample some training data $\mathcal{D}^{(t)}$, (iii) form the posterior predictive $q_\upsilon(\cdot \mid \mathcal{D}^{(t)})$ and, (iv) compute the log-density $\log q_\upsilon(y^{*(t)} \mid \mathcal{D}^{(t)})$ at test data $y^{*(t)}$ *not included in* $\mathcal{D}^{(t)}$. Repeating this process many times and averaging the results provides an unbiased estimate of the objective which can then be optimised with respect to the global parameters φ, but also the variational parameters υ. This perspective also makes it clear that the procedure is scoring the approximate inference method by simulating approximate Bayesian held-out log-likelihood evaluation.

Importantly, while an inference network is used to approximate posterior distributions, the training procedure differs significantly from standard variational inference. In particular, rather than minimising $\mathrm{KL}(q_\upsilon(\psi \mid \mathcal{D}) \parallel p(\psi \mid \mathcal{D}))$, our objective function directly focuses on the posterior predictive distribution and minimises $\mathrm{KL}(p(y^* \mid \mathcal{D}) \parallel q_\upsilon(y^* \mid \mathcal{D}))$.

We explain how to turn the objective function into a practical end-to-end stochastic training objective in Section 8.1.4. In the next section we provide a derivation of our approach that is grounded in Bayesian decision theory.

8.1.3 *Bayesian decision theoretic generalisation of ML-PIP*

A generalisation of the new inference framework presented above is based upon Bayesian decision theory (BDT). BDT provides a recipe for making predictions $\widehat{y}$

1 Note that the true *predictive* posterior $p(y \mid \mathcal{D})$ is recovered regardless of the identifiability of $p(\psi \mid \mathcal{D})$.

for an unknown test variable y^* by combining information from observed training data $\mathcal{D}^{(t)}$ (here from a single task t) and a loss function $L(y^*, \widehat{y})$ that encodes the cost of predicting $\widehat{y}$ when the true value is y^* (Berger, 2013; Jaynes, 2003). In BDT an optimal prediction minimises the expected loss[2]:

$$\widehat{y}^* = \underset{\widehat{y}}{\operatorname{argmin}} \int p(y^* \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi) L(y^*, \widehat{y}) \mathrm{d}y^* \,, \tag{8.7}$$

where

$$p(y^* \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi) = \int p(y^* \mid \mathbf{x}^*, \psi^{(t)}, \varphi) p(\psi^{(t)} \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi) \, \mathrm{d}\psi^{(t)} \tag{8.8}$$

is the Bayesian predictive distribution and $p(\psi^{(t)} \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi)$ the posterior distribution of $\psi^{(t)}$ given the training data from task t.

BDT naturally separates test and training data and so is a natural lens through which to view recent episodic training approaches as introduced in Section 6.1. Based on this insight, what follows is a derivation of a stochastic variational objective for meta-learning probabilistic inference. Although the derivation is fairly dense, it is ultimately simple and rigorously grounded in Bayesian inference and decision theory.

Distributional BDT.

We generalise BDT to cases where the goal is to return a full predictive *distribution* $q(\cdot)$ over the unknown test variable y^* rather than a point prediction. The quality of q is quantified through a distributional loss function $L(y^*, q(\cdot))$. Typically, if y^* (the true value of the underlying variable) falls in a low probability region of $q(\cdot)$ the loss will be high, and vice versa. The optimal predictive q^* is found by optimising the expected distributional loss with q constrained to a family $\mathcal{Q}$:

$$q^* = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \int p(y^* \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi) L(y^*, q(\cdot)) \mathrm{d}y^*. \tag{8.9}$$

Amortised variational training.

Here, we amortise q to form quick predictions at test time and learn parameters by minimising the average expected loss *over tasks*. Let υ be a set of shared variational parameters such that $q(y^*) = q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi)$ (or q_υ for short). With these assumptions, the approximate predictive distribution can take any training dataset $\mathcal{D}^{(t)}$ as an argument and directly perform prediction of $y^{*(t)}$. The optimal

2 For discrete outputs the integral may be replaced with a summation.

variational parameters are found by minimising the expected distributional loss across tasks

$$\begin{aligned} \upsilon^* &= \arg\min_{\upsilon} \mathcal{L}\left[q_\upsilon\right] \\ \mathcal{L}\left[q_\upsilon\right] &= \iint p(\mathcal{D}^{(t)}) p(y^* \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi) L(y^*, q_\upsilon(\cdot \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi)) \,\mathrm{d}y^* \,\mathrm{d}\mathcal{D}^{(t)} \\ &\approx \mathbb{E}_{p(\mathcal{D}^{(t)}, y^*)}\left[L(y^*, q_\upsilon(\cdot \mid \mathbf{x}^*, \mathcal{D}^{(t)}, \varphi))\right]. \end{aligned} \tag{8.10}$$

Here the variables $\mathcal{D}^{(t)}, \mathbf{x}^*$ and y^* are placeholders for integration over all possible datasets, test inputs, and test outputs. Note that Equation (8.10) can be stochastically approximated by sampling a task t and randomly partitioning into training data $\mathcal{D}$ and test data $\{\mathbf{x}_m^*, y_m^*\}_{m=1}^M$, which naturally recovers episodic mini-batch training over tasks and data (Vinyals et al., 2016; Ravi and Larochelle, 2017). Critically, this does not require computation of the true predictive distribution. It also emphasises the meta-learning aspect of the procedure, as the model is learning how to infer predictive distributions from training tasks.

Loss functions.

As distributional loss function L, we employ the convenient log-loss, that is, the negative log density of q_υ at y^*. In this case,

$$\mathcal{L}\left[q_\upsilon\right] = \mathbb{E}_{p(\mathcal{D}, y^*)}\left[-\log q_\upsilon(y^* \mid \mathcal{D})\right] \tag{8.11}$$

$$= \mathbb{E}_{p(\mathcal{D})}\left[\mathrm{KL}(p(y^* \mid \mathcal{D}) \parallel q_\upsilon(y^* \mid \mathcal{D})) + \mathrm{H}\left[p(y^* \mid \mathcal{D})\right]\right], \tag{8.12}$$

where $\mathrm{H}\left[p(y)\right]$ is the entropy of p. Equation (8.11) has the elegant property that the optimal q_υ is the closest member of $\mathcal{Q}$ (in a KL sense) to the true predictive $p(y^* \mid \mathcal{D})$, which is unsurprising as the log-loss is a proper scoring rule (Huszar, 2013). This is reminiscent of the sleep phase in the wake-sleep algorithm (Hinton et al., 1995). Exploration of alternative proper scoring rules (Dawid, 2007) and more task-specific losses (Lacoste–Julien et al., 2011) is left for future work.

8.1.4 *End-to-end stochastic training objective*

Armed by the insights from Section 8.1 we now describe the full end-to-end training procedure for the objective Equation (8.6):

$$\mathcal{L}\left(\upsilon, \varphi\right) = -\mathbb{E}_{p(\mathcal{D}, y^*, \mathbf{x}^*)}\left[\log q_\upsilon(y^* \mid \mathbf{x}^*, \mathcal{D}, \varphi)\right] \tag{8.13}$$

$$= -\mathbb{E}_{p(\mathcal{D}, y^*, \mathbf{x}^*)}\left[\log \int p(y^* \mid \mathbf{x}^*, \psi, \varphi) q_\upsilon(\psi \mid \mathcal{D}, \varphi) \,\mathrm{d}\psi\right]. \tag{8.14}$$

We optimise the objective over the shared parameters φ as this will maximise predictive performance, that is, the Bayesian held-out likelihood. An end-to-end stochastic training objective for υ and φ is:

$$\widehat{\mathcal{L}}(\upsilon, \varphi) = \frac{1}{MT} \sum_{M,T} \log \frac{1}{L} \sum_{l=1}^{L} p\left(y_m^{*(t)} \mid \mathbf{x}_m^{*(t)}, \psi_l^{(t)}, \varphi\right), \tag{8.15}$$
$$\text{with } \psi_l^{(t)} \sim q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \varphi) \text{ and } \{y_m^{*(t)}, \mathbf{x}_m^{*(t)}, \mathcal{D}^{(t)}\} \sim p_{\text{Data}}(y^*, \mathbf{x}^*, \mathcal{D}^{(t)})$$

where p_{Data} represents the data distribution; for example, sampling tasks t and splitting them into disjoint training data $\mathcal{D}^{(t)}$ and test data $\{(\mathbf{x}_m^{*(t)}, y_m^{*(t)})\}_{m=1}^{M_t}$. Therefore, as discussed above, training uses episodic train/test splits at meta-training time. We have also approximated the integral over ψ using L Monte Carlo samples. The local reparametrisation trick (Kingma et al., 2015) enables optimisation. Interestingly, the learning objective does not require an *explicit* specification of the prior distribution over parameters, $p(\psi^{(t)} \mid \varphi)$. Instead, it is learned *implicitly* through $q_\upsilon(\psi \mid \mathcal{D}, \varphi)$.

In summary, we have developed an approach for Meta-Learning Probabilistic Inference for Prediction (ML-PIP). A simple investigation of the inference method with synthetic data is provided in Section 8.4.1. In Section 8.3 we show that this formulation unifies a number of existing approaches, but first we discuss a particular instance of the ML-PIP framework that supports versatile learning.

8.2 VERSATILE AMORTISED INFERENCE

A versatile system is one that makes inferences both rapidly *and* flexibly. By "rapidly" we mean that test-time inference should involve only simple computations such as feed-forward passes through a neural network. By "flexibly" we mean that the system should support a variety of tasks – including variable numbers of shots or numbers of classes in classification problems – without retraining. Rapid inference comes automatically with the use of a deep neural network to amortise the approximate posterior distribution q_υ. However, it typically comes at the cost of flexibility: amortised inference is usually limited to a single specific task. Below, we discuss two design choices that enable us to retain flexibility: (i) amortisation of sets and (ii) a context independence assumption.

8.2.1 *Inference with sets as inputs*

Amortisation networks employed in VAEs map a single datapoint $\mathbf{x}$ to a distribution in the latent space $q(\mathbf{z} \mid \mathbf{x})$. In contrast, our amortisation network $q(\psi \mid \mathcal{D}, \mathbf{x}, \varphi)$ should handle *datasets* of variable sizes $\mathcal{D}$ as inputs and should be invariant to the

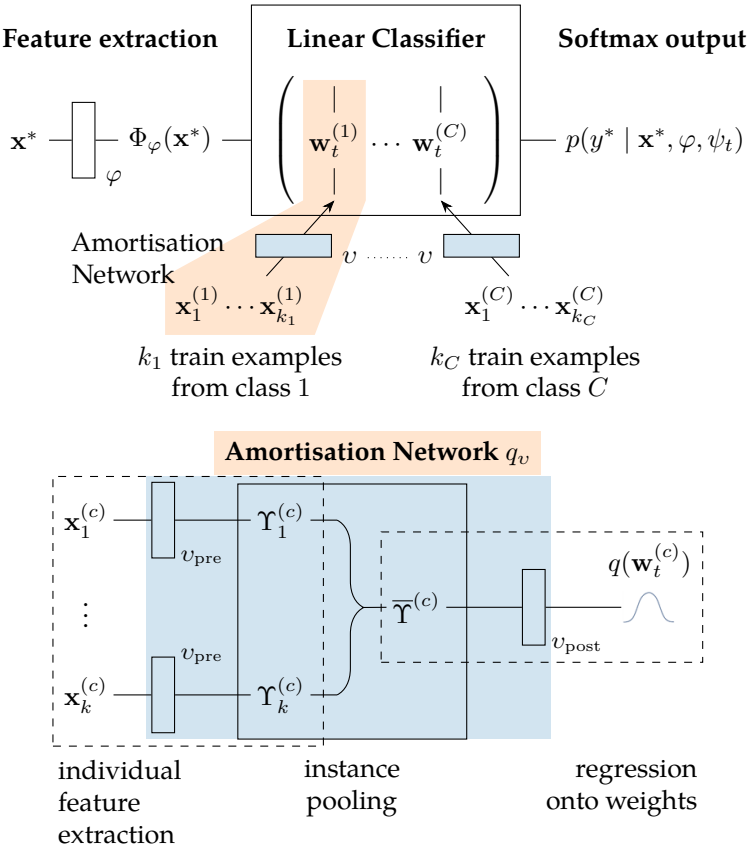


Figure 8.2: Computational flow of VERSA for few-shot classification with the context-independent approximation. *Top:* A test point $\mathbf{x}^*$ is mapped to its softmax output through a feature extractor neural network and a linear classifier (fully connected layer). The global parameters φ of the feature extractor are shared between tasks whereas the weight vectors $\mathbf{w}_t^{(c)}$ of the linear classifier are task specific and inferred through an amortisation network with parameters φ. *Bottom:* Amortisation network that maps the extracted features of the k training examples of a particular class to the corresponding weight vector of the linear classifier.

ordering of these sets. We therefore use permutation-invariant *instance-pooling* operations to process these sets similarly to Qi et al. (2017) and as formalised in Zaheer et al. (2017). The instance-pooling operation also ensures that the network can process any number of training observations, see Figure 8.2 *(bottom)*.

8.2.2 *VERSA for few-shot image classification*

For few-shot image classification, our parameterisation of the probabilistic model is inspired by early work from Heskes (2000) and Bakker and Heskes (2003) and recent extensions to deep learning (Bauer* et al., 2017b; Qiao et al., 2018). A feature extractor neural network $\Phi_\varphi(x) \in \mathbb{R}^{d_\varphi}$, shared across all tasks, feeds into a set

of task-specific linear classifiers with softmax outputs and weights and biases $\psi^{(t)} = \{W^{(t)}, \mathbf{b}^{(t)}\}$ (see Figure 8.2 *(top)*).

A naive amortisation scheme requires the approximate posterior $q_\upsilon(\psi \mid \mathcal{D}, \varphi)$ to model the distribution over full weight matrices in $\mathbb{R}^{d_\varphi \times C}$ (and biases). This requires the specification of the number of few-shot classes C ahead of time and limits inference to this chosen number. Moreover, it is difficult to meta-learn systems that directly output large matrices as the output dimensionality is high. We therefore propose specifying $q_\upsilon(\psi \mid \mathcal{D}, \varphi)$ in a *context independent* manner such that each weight vector ψ_c depends only on examples from class c, by amortising individual weight vectors associated with a single softmax output instead of the entire weight matrix directly. To reduce the number of learned parameters, the amortisation network operates directly on the extracted features $\Phi_\varphi(\mathbf{x})$ and not on the inputs $\mathbf{x}$, that is, we tie υ_{pre} to φ in Figure 8.2 such that $\Upsilon_i^{(c)}(\mathbf{x}_i) = \Phi_\varphi(\mathbf{x}_i)$:

$$q_\upsilon(\psi \mid \mathcal{D}, \varphi) = \prod_{c=1}^{C} q_\upsilon\left(\psi_c \mid \{\Phi_\varphi\left(\mathbf{x}_n^c\right)\}_{n=1}^{k_c}, \varphi\right). \tag{8.16}$$

Note that in our implementation, end-to-end training is employed; we back-propagate to υ through the inference network. Here k_c is the number of observed examples in class c and $\psi_c = \{\mathbf{w}_c, b_c\}$ denotes the weight vector and bias of the linear classifier associated with that class. Thus, we construct the classification matrix $\psi^{(t)}$ by performing C feed-forward passes through the inference network $q_\upsilon(\psi \mid \mathcal{D}, \varphi)$, see Figure 8.2.

The assumption of context independent inference is an approximation that we also employed in our previous approach in Chapter 7. In Appendix C.1, we provide theoretical and empirical justification for its validity. Our theoretical arguments use insights from Density Ratio Estimation (Mohamed, 2018; Sugiyama et al., 2012), and we empirically demonstrate that full approximate posterior distributions are close to their context independent counterparts.

Critically, the context independent approximation addresses all the limitations of a naive amortisation mentioned above: (i) the inference network needs to amortise far fewer parameters whose number does not scale with number of classes C (a single weight vector instead of the entire matrix); (ii) the amortisation network can be meta-trained with different numbers of classes per task, and (iii) the number of classes C can vary at test-time. By limiting the complexity of the model through the context independent assumption, we therefore provide inductive biases that enable the model to be both fast and flexible while achieving good performance with a low number of examples.

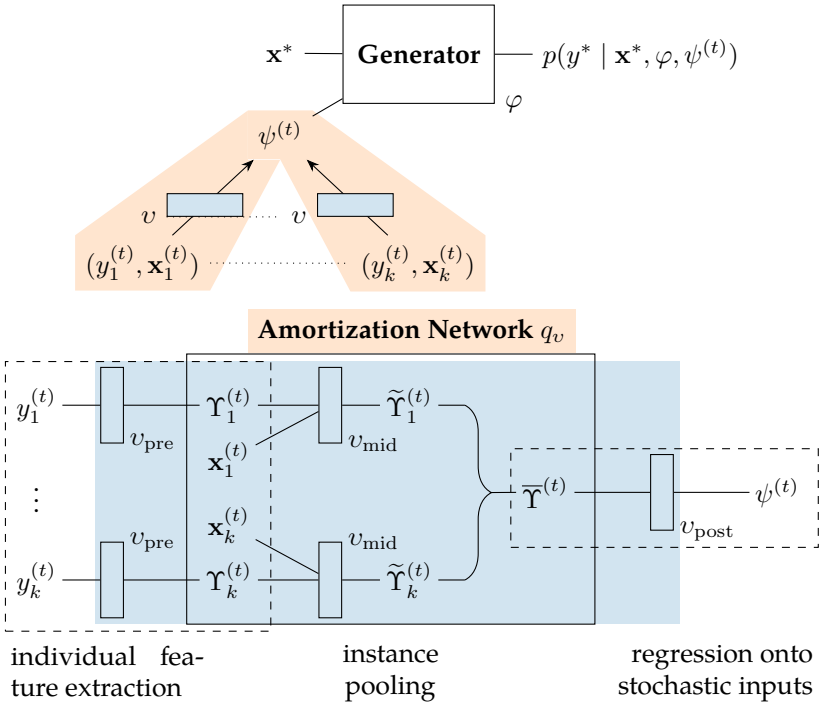


Figure 8.3: Computational flow of VERSA for few-shot view reconstruction. *Top:* A set of training images and angles $\{(y_n^{(t)}, x_n^{(t)})\}_{n=1}^k$ are mapped to a stochastic input $\psi^{(t)}$ through the amortisation network q_υ. $\psi^{(t)}$ is then concatenated with a test angle $\mathbf{x}^*$ and mapped onto a new image through the generator φ. *Bottom:* Amortisation network that maps k image/angle examples of a particular object-instance to the corresponding stochastic input.

8.2.3 *VERSA for few-shot image reconstruction (regression)*

To showcase that a variant of VERSA can be used for regression, we consider a challenging few-shot learning task with a complex (high dimensional and continuous) output space. We define view reconstruction as the ability to infer how an object looks from any desired angle based on a small set of observed views. We frame this as a multi-output regression task from a set of training images with known orientations to output images with specified orientations.

Our generative model is similar to the generator of a GAN or the decoder of a VAE: A latent vector $\psi^{(t)} \in \mathbb{R}^{d_\psi}$, which acts as an object-instance level input to the generator, is concatenated with an angle representation and mapped through the generator to produce an image at the specified orientation. In this setting, we treat all parameters φ of the generator network as global parameters (see Appendix C.3.1 for full details of the architecture), whereas the latent inputs $\psi^{(t)}$ are the task-specific parameters. We use a Gaussian likelihood in pixel space for the outputs of the generator. To ensure that the output means are between zero and one, we use a sigmoid activation after the final layer. υ parameterises an amortisation network that first processes the image representations of an object,

concatenates them with their associated view orientations, and processes them further before instance-pooling. From the pooled representations, $q_v(\psi \mid \mathcal{D}, \varphi)$ produces a distribution over vectors $\psi^{(t)}$. This process is illustrated in Figure 8.3.

8.3 ML-PIP UNIFIES DISPARATE RELATED APPROACHES TO FEW-SHOT LEARNING

In this section, we continue in the spirit of Grant et al. (2018), and recast a broader class of meta-learning approaches as approximate inference in hierarchical models. We show that ML-PIP unifies a number of important approaches to meta-learning introduced in Section 6.4, including gradient and metric based variants, as well as amortised MAP inference and conditional modelling approaches (Garnelo et al., 2018a). We lay out these connections, most of which rely on point estimates for the task-specific parameters corresponding to

$$q(\psi^{(t)} \mid \mathcal{D}^{(t)}, \varphi) = \delta\left(\psi^{(t)} - \psi^*(D^{(t)}, \varphi)\right) . \tag{8.17}$$

In addition, we compare previous approaches to VERSA.

8.3.1 *Gradient-based meta-learning*

Let the task-specific parameters $\psi^{(t)}$ be all the parameters in a neural network. Consider a point estimate formed by taking a gradient ascent step of the training loss, initialised at ψ_0 and with learning rate η.

$$\psi^*(D^{(t)}, \varphi) = \psi_0 + \eta \frac{\partial}{\partial \psi} \sum_{n=1}^{N_t} \log p(y_n^{(t)} \mid x_n^{(t)}, \psi, \varphi)\Big|_{\psi_0} . \tag{8.18}$$

This is an example of semi-amortised inference (Kim et al., 2018), as the only shared inference parameters are the initialistion and learning rate, and optimisation is required for each task (albeit only for one step). Importantly, Equation (8.18) recovers *Model-agnostic meta-learning* (MAML) (Finn et al., 2017), providing a perspective as semi-amortised ML-PIP. This perspective is complementary to that of Grant et al., 2018 who justify the one-step gradient parameter update employed by MAML through MAP inference and the form of the prior $p(\psi \mid \varphi)$. Note that the episodic train/test splits do not follow naturally from this perspective. Instead we view the update choice as one of amortisation which is trained using the predictive KL and naturally recovers the test-train splits. More generally, multiple gradient steps could be fed into an RNN to compute ψ^* which recovers Ravi and Larochelle (2017). In comparison to these methods, besides being distributional over ψ, VERSA relieves the need to back-propagate through gradient based updates

during training or compute gradients at test time. Moreover, it also enables the treatment of both local and global parameters which simplifies inference.

8.3.2 *Metric-based few-shot learning*

Let the task-specific parameters be the top layer softmax weights and biases of a neural network $\psi^{(t)} = \{\mathbf{w}_c^{(t)}, b_c^{(t)}\}_{c=1}^C$. The shared parameters are the lower layer weights. Consider amortised point estimates for these parameters constructed by averaging the top-layer activations for each class,

$$\begin{aligned} \psi^*(\mathcal{D}^{(t)}, \varphi) = \{\mathbf{w}_c^*, b_c^*\}_{c=1}^C = \left\{\mu_c^{(t)}, -\|\mu_c^{(t)}\|^2/2\right\}_{c=1}^C \\ \text{where } \mu_c^{(t)} = \frac{1}{kC} \sum_{n=1}^{N=k\cdot C} \Phi_\varphi(\mathbf{x}_n^{(c)}) \end{aligned} \tag{8.19}$$

These choices lead to the following predictive distribution:

$$\begin{aligned} p(y^{*(t)} = c \mid \mathbf{x}^{*(t)}, \varphi) &\propto \exp\left(-d(\Phi_\varphi(\mathbf{x}^{*(t)}), \mu_c^{(t)})\right) \\ &= \exp\left(\Phi_\varphi(\mathbf{x}^{*(t)})^T \mu_c^{(t)} - \frac{1}{2}\|\mu_c^{(t)}\|^2\right), \end{aligned} \tag{8.20}$$

which recovers Prototypical Networks (Snell et al., 2017) using a Euclidean distance function d with the final hidden layer being the embedding space, compare to Equation (6.3). In comparison, VERSA is distributional and it uses a more flexible amortisation function that goes beyond averaging of activations.

8.3.3 *Amortised MAP inference*

Qiao et al. (2018) proposed a method for predicting weights of classes from activations of a pre-trained network to support (i) online learning on a single task to which new few-shot classes are incrementally added, (ii) transfer from a high-shot classification task to a separate low-shot classification task. This is an example usage of hyper-networks (Ha et al., 2017) to amortise learning about weights, and can be recovered by the ML-PIP framework by pre-training φ and performing MAP inference for ψ. Our transfer approach in Chapter 7 also uses pre-trained features Φ_φ but optimises ψ instead of using amortisation; it is similarly recovered by ML-PIP. VERSA goes beyond point estimates and although its amortisation network is similar in spirit, it is more general, employing end-to-end training and supporting full multi-task learning by sharing information between many tasks.

8.3.4 *Conditional models trained via maximum likelihood*

In cases where a point estimate of the task-specific parameters are used the predictive becomes

$$q_\upsilon(y^* \mid \mathcal{D}, \varphi) = \int p(y^* \mid \psi, \varphi) q_\upsilon(\psi \mid \mathcal{D}, \varphi) \,\mathrm{d}\psi = p(y^* \mid \psi^*(\mathcal{D}, \varphi), \varphi). \tag{8.21}$$

In such cases the amortisation network that computes $\psi^*(\mathcal{D}, \varphi)$ can be equivalently viewed as part of the model specification rather than the inference scheme. From this perspective, the ML-PIP training procedure for υ and φ is equivalent to training a conditional model $p(y^* \mid \psi^*_\upsilon(\mathcal{D}, \varphi), \varphi)$ via maximum likelihood estimation, establishing a strong connection to neural processes (Garnelo et al., 2018a,b).

8.3.5 *Comparison to (standard) variational inference*

Instead of our objective function, we can also apply standard variational inference to the task specific parameters ψ in the multi-task discriminative model. Using the same ansatz as for the approximate posterior in Equation (8.3), we can treat the variational parameters υ of $q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \varphi)$ in an amortised or non-amortised fashion. In the latter case, we need to optimise local variational parameters for each task separately instead of training an amortisation network. However, the derivation of the corresponding evidence lower bound (ELBO) objective is the same.

For a single task t, the ELBO may be expressed as:

$$\begin{aligned} \mathcal{L}_t = \mathbb{E}_{q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \varphi)} \left[\sum_{(\mathbf{x}, y) \in \mathbb{D}^{(t)}} \log p(y \mid \mathbf{x}, \psi, \varphi) \right] \\ - \mathrm{KL}(q_\phi(\psi \mid \mathcal{D}^{(t)}, \varphi) \parallel p(\psi \mid \varphi)) \,, \end{aligned} \tag{8.22}$$

where $p(\psi \mid \varphi)$ is an *explicit* prior on the task-specific parameters. We can then derive a stochastic estimator to optimise Equation (8.22) by sampling $\mathcal{D}^{(t)} \sim p(\mathcal{D})$ (approximated with a training set of tasks) and simple Monte Carlo integration over ψ such that $\psi^{(l)} \sim q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \theta)$:

$$\begin{aligned} \hat{\mathcal{L}}(\upsilon, \varphi) = \frac{1}{T} \sum_{t=1}^{T} \Bigg(\sum_{(\mathbf{x}, y) \in \mathcal{D}^{(t)}} \left(\frac{1}{L} \sum_{l=1}^{L} \log p(y^{(t)} \mid \mathbf{x}^{(t)}, \psi_l^{(t)}, \varphi) \right) \\ - \mathrm{KL}(q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \varphi) \parallel p(\psi \mid \varphi)) \Bigg) \,, \end{aligned} \tag{8.23}$$

where $\psi_l^{(t)} \sim q_\upsilon(\psi \mid \mathcal{D}^{(t)}, \varphi)$. In addition to the conceptual difference from ML-PIP (discussed in Section 8.1.1), this differs from the ML-PIP objective by (i) not employing meta train / test splits, and (ii) including the KL for regularisation instead. In Section 8.4, we show that VERSA significantly improves over standard VI in the few-shot classification case and compare to recent VI/meta-learning hybrids.

8.4 EMPIRICAL EVALUTION

We evaluate VERSA on several few-shot learning tasks. We begin with illustrative toy experiments to investigate the properties of the amortised posterior inference achieved by VERSA. We then report few-shot classification results using the Omniglot and *mini*ImageNet datasets in Section 8.4.2, and demonstrate VERSA's ability to retain high accuracy as the shot and way are varied at test time. In Section 8.4.3, we examine VERSA's performance on a one-shot view reconstruction task with ShapeNet objects. Source code for the experiments is available at `https://github.com/Gordonjo/versa`.

8.4.1 *Posterior inference with illustrative toy data*

To investigate the approximate inference performed by our training procedure, we generate data from a Gaussian distribution with means varying across tasks:

$$\begin{aligned} p(\varphi) &= \delta(\varphi - 0) \\ p\left(\psi^{(t)} \mid \varphi\right) &= \mathcal{N}\left(\psi^{(t)}; \varphi, \sigma_\psi^2\right) \\ p\left(y_n^{(t)} \mid \psi^{(t)}\right) &= \mathcal{N}\left(y_n^{(t)}; \psi^{(t)}, \sigma_y^2\right). \end{aligned} \tag{8.24}$$

We generate $T = 250$ tasks in two separate experiments with $N \in \{5, 10\}$ train observations and $M = 15$ test observations. We introduce a very simple single-layer inference network $q_\upsilon(\psi \mid \mathcal{D}^{(t)}) = \mathcal{N}(\psi; \mu_q^{(t)}, \sigma_q^{(t)2})$, with:

$$\begin{aligned} \mu_q^{(t)} &= w_\mu \sum_{n=1}^{N} y_n^{(t)} + b_\mu \\ \sigma_q^{(t)2} &= \exp\left(w_\sigma \sum_{n=1}^{N} y_n^{(t)} + b_\sigma\right). \end{aligned} \tag{8.25}$$

The learnable parameters $\upsilon = \{w_\mu, b_\mu, w_\sigma, b_\sigma\}$ are trained with the objective function in Equation (8.15). The model is trained to convergence with Adam (Kingma and Ba, 2015) using mini-batches of tasks from the generated dataset. Then, a separate set of tasks is generated from the same generative process, and

the posterior $q_\upsilon(\psi \mid \mathcal{D})$ is inferred with the learned amortisation parameters υ. The true posterior over ψ is Gaussian with a mean that depends on the task, and may be computed analytically. Figure 8.4 shows the approximate posterior distributions inferred for unseen test sets by the trained amortisation networks (——) compared to the true posteriors (——). The evaluation shows that the inference procedure is able to recover accurate posterior distributions over ψ, despite minimising a predictive KL divergence in data space.

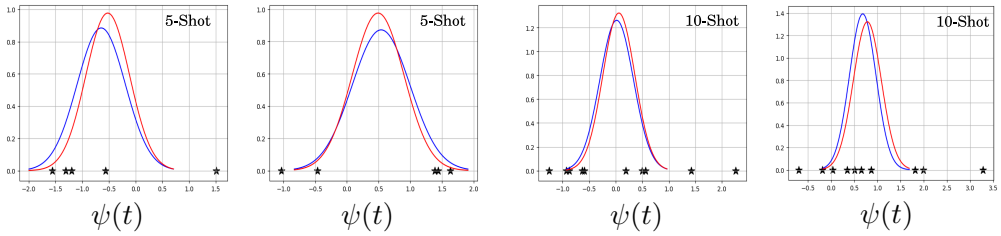


Figure 8.4: Comparison of true posteriors $p(\psi \mid \mathcal{D})$ (——) and amortised approximate posteriors $q_\upsilon(\psi \mid \mathcal{D})$ (——) for unseen test tasks ($\star$) in the experiment. We show four examples, two for 5-shot learning and two for 10-shot learning. In all cases, the approximate posterior closely resembles the true posterior given the observed data.

8.4.2 *Few-shot classification benchmarks*

We evaluate VERSA on standard few-shot classification tasks in comparison to previous work. Specifically, we consider the Omniglot (Lake et al., 2011) and *mini*ImageNet (Ravi and Larochelle, 2017) datasets which are C-way classification tasks with k_c examples per class. VERSA follows the implementation in Sections 8.1 and 8.2, and the approximate inference scheme in Equation (8.16). We follow the experimental protocol established by Vinyals et al. (2016) for Omniglot and Ravi and Larochelle (2017) for *mini*Imagenet, using equivalent architectures for Φ_φ. Training is carried out in an episodic manner: for each task, k_c examples are used as training inputs to infer $q_\upsilon(\psi^{(c)} \mid \mathcal{D}, \varphi)$ for each class, and an additional set of examples is used to evaluate the objective function. Full details of data preparation and network architectures are provided in Appendix C.2. For Omniglot, the training, validation, and test splits have not been specified for previous methods, which affects the comparison.

Table 8.1 details few-shot classification performance for VERSA as well as competitive approaches. The tables include results for only those approaches with comparable training procedures and convolutional feature extraction architectures. Approaches that employ pre-training and/or residual networks such as our previous approach (Chapter 7) or (Qiao et al., 2018; Rusu et al., 2019; Gidaris and Komodakis, 2018; Oreshkin et al., 2018; Satorras and Estrach, 2018; Lacoste et al.,

2018) have been excluded so that the quality of the learning algorithm can be assessed separately from the power of the underlying feature extractor.

VERSA achieves a new state-of-the-art results (67.37% – up 1.38% over the previous best) on 5-way-5-shot classification on the *mini*ImageNet benchmark and (97.66% – up 0.02%) on the 20-way-1-shot Omniglot benchmark for systems using a convolution-based network architecture and an end-to-end training procedure. VERSA is within error bars of state-of-the-art on three other benchmarks including 5-way-1-shot *mini*ImageNet, 5-way-5-shot Omniglot, and 5-way-1-shot Omniglot. Results on the Omniglot 20-way-5-shot benchmark are very competitive with, but lower than, other approaches. While most of the methods evaluated in Table 8.1 adapt all of the learned parameters for new tasks, VERSA is able to achieve state-of-the-art performance despite adapting only the weights of the top-level classifier.

Comparison to non-amortised and amortised standard variational inference.

To investigate the performance of our inference procedure, we compare it in terms of log-likelihood (Table 8.2) and accuracy (Table 8.1) to training the same model using both amortised and non-amortised standard variational inference (VI). That is, instead of our objective (Equation (8.13)) we use the standard VI objective (Equation (8.23)) as discussed in Section 8.3.5. VERSA improves substantially over amortised VI even though the same amortisation network is used for both. This is due to VI's tendency to underfit, especially for small numbers of data points (Turner and Sahani, 2011; Trippe and Turner, 2018), which is compounded when using inference networks (Cremer et al., 2018). Using non-amortised VI improves performance over amortised VI, but does not reach the level of VERSA. Moreover, forming the posterior is significantly slower as it requires many forward/backward passes through the network to optimise the task-specific parameters. This is similar in spirit to MAML (Finn et al., 2017), though MAML dramatically reduces the number of required iterations by finding good global initialisations. For example, MAML uses only five gradient steps for *mini*ImageNet. This is in contrast to the single forward pass required by VERSA.

Versatility.

VERSA allows us to vary the number of classes C and shots k_c between training and testing (Equation (8.16)). Figure 8.5 *(left)* shows that a model trained for a particular C-way retains very high accuracy as C is varied. For example, when VERSA is trained for the 20-way-5-shot condition, at test-time it can handle $C = 100$-way conditions and retain an accuracy of approximately 94%. Figure 8.5 *(right)* shows similar robustness as the number of shots k_c is varied. VERSA therefore demonstrates considerable flexibility and robustness to the test-time conditions,

	Omniglot				*miniImageNet*	
	5-way accuracy (%)		**20-way accuracy (%)**		**5-way accuracy (%)**	
Method	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Siamese Nets (Koch et al., 2015)	97.3	98.4	88.1	97.0		
Matching Nets (Vinyals et al., 2016)	98.1	98.9	93.8	98.5	46.6	60.0
Neural Statistician (Edwards and Storkey, 2017)	98.1	99.5	93.2	98.1		
Memory Mod (Kaiser et al., 2017)	98.4	99.6	95.0	98.6		
Meta LSTM (Ravi and Larochelle, 2017)					43.44 ± 0.77	60.60 ± 0.71
MAML (Finn et al., 2017)	98.7 ± 0.4	**99.9 ± 0.1**	95.8 ± 0.3	98.9 ± 0.2	48.7 ± 1.84	63.11 ± 0.92
Prototypical Nets[3] (Snell et al., 2017)	97.4	99.3	95.4	98.7	46.61 ± 0.78	65.77 ± 0.70
mAP-SSVM (Triantafillou et al., 2017)	98.6	99.6	95.2	98.6	50.32 ± 0.80	63.94 ± 0.72
mAP-DLM (Triantafillou et al., 2017)	98.8	99.6	95.4	98.6	50.28 ± 0.80	63.70 ± 0.70
LLAMA (Grant et al., 2018)					49.40 ± 1.83	
PLATIPUS (Finn et al., 2018)					50.13 ± 1.86	
Meta-SGD (Li et al., 2017)	**99.53 ± 0.26**	**99.93 ± 0.09**	95.93 ± 0.38	98.97 ± 0.19	50.47 ± 1.87	64.03 ± 0.94
SNAIL (Mishra et al., 2018)	99.07 ± 0.16	99.78 ± 0.09	**97.64 ± 0.30**	**99.36 ± 0.18**	45.1	55.2
Relation Net (Sung et al., 2018)	**99.6 ± 0.2**	**99.8 ± 0.1**	**97.6 ± 0.2**	**99.1 ± 0.1**	50.44 ± 0.82	65.32 ± 0.70
Reptile (Nichol and Schulman, 2018)	97.68 ± 0.04	99.48 ± 0.06	89.43 ± 0.14	97.12 ± 0.32	49.97 ± 0.32	**65.99 ± 0.58**
BMAML (Yoon et al., 2018)					**53.8 ± 1.46**	
Amortised VI	97.77 ± 0.55	98.71 ± 0.22	90.56 ± 0.54	96.12 ± 0.23	44.13 ± 1.78	55.68 ± 0.91
Non-Amortised VI	98.77 ± 0.18	99.74 ± 0.06	95.28 ± 0.19	98.84 ± 0.09		
VERSA (Ours)	**99.70 ± 0.20**	**99.75 ± 0.13**	**97.66 ± 0.29**	98.77 ± 0.18	**53.40 ± 1.82**	**67.37 ± 0.86**

Table 8.1: Accuracy results for different few-shot settings on Omniglot and *mini*ImageNet. The ± sign indicates the 95% confidence interval over tasks using a Student's t-distribution approximation. Bold text indicates the highest scores that overlap in their confidence intervals.

	Omniglot			
	5-way NLL		**20-way NLL**	
Method	1-shot	5-shot	1-shot	5-shot
Amortised VI	0.179 ± 0.009	0.137 ± 0.004	0.456 ± 0.010	0.253 ± 0.004
Non-Amortised VI	0.144 ± 0.005	0.025 ± 0.001	0.393 ± 0.005	0.078 ± 0.002
VERSA	0.010 ± 0.005	0.007 ± 0.003	0.079 ± 0.009	0.031 ± 0.004

	miniImageNet	
	5-way NLL	
Method	1-shot	5-shot
Amortised VI	1.328 ± 0.024	1.165 ± 0.010
VERSA	1.183 ± 0.023	0.859 ± 0.015

Table 8.2: Negative log-likelihood (NLL) results for different few-shot settings on Omniglot (top) and *mini*ImageNet (bottom). The $\pm$ sign indicates the 95% confidence interval over tasks using a Student's t-distribution approximation.

but at the same time it is efficient as it only requires forward passes through the network. The time taken to evaluate 1000 test tasks with a 5-way-5-shot *mini*ImageNet trained model using MAML[4] is 302.9 seconds whereas VERSA took 53.5 seconds on a NVIDIA Tesla P100-PCIE-16GB GPU. This is more than a $5\times$ speed advantage in favor of VERSA while bettering MAML in accuracy by 4.26%.

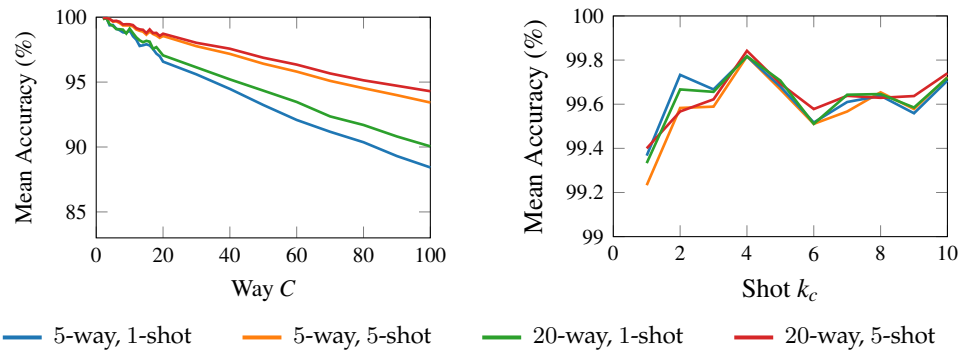


Figure 8.5: Test accuracy on Omniglot when varying way (fixing shot to be that used for training) *(left)* and shot *(right)*. In *(left)*, all models are evaluated on 5-way classification. Colors indicate models trained with different way-shot episodic combinations.

8.4.3 *ShapeNet view reconstruction*

ShapeNetCore v2 (Chang et al., 2015) is a database of 3D objects covering 55 common object categories with ~51,300 unique objects. For our experiments, we

4 code obtained from https://github.com/cbfinn/maml

use 12 of the largest object categories. We concatenate all instances from all 12 of the object categories together to obtain a dataset of 37,108 objects. This dataset is then randomly shuffled and we use 70% of the objects for training, 10% for validation, and 20% for testing. For each object, we generate 36 views of size 32×32 pixels spaced evenly every 10 degrees in azimuth around the object.

We evaluate VERSA by comparing it to a conditional variational autoencoder (C-VAE) with view angles as labels (Kingma et al., 2014; Narayanaswamy et al., 2017) and identical architectures. We train VERSA in an episodic manner and the C-VAE in batch-mode on all 12 object classes at once. We train on a single view selected at random and use the remaining views to evaluate the objective function. For full experimentation details see Appendix C.3. Figure 8.6 shows views of unseen objects from the test set generated from a single shot with VERSA as well as a C-VAE and compares both to ground truth views. Both VERSA and the C-VAE capture the correct orientation of the object in the generated images. However, VERSA produces images that contain much more detail and are visually sharper than the C-VAE images. Although important information is missing due to occlusion in the single shot, VERSA is often able to accurately impute this information presumably due to learning the statistics of these objects. Table 8.3 provides quantitative comparison results between VERSA with varying shot and the C-VAE. The quantitative metrics all show the superiority of VERSA over a C-VAE. As the number of shots increase to 5, the measurements show a corresponding improvement.

Model	MSE	SSIM
C-VAE 1-shot	0.0269	0.5705
VERSA 1-shot	0.0108	0.7893
VERSA 5-shot	0.0069	0.8483

Table 8.3: View reconstruction test results. Mean squared error (MSE – lower is better) and the structural similarity index (SSIM – higher is better) (Wang et al., 2004) are measured between the generated and ground truth images. Error bars not shown as they are insignificant.

8.5 DISCUSSION

VERSA appears very similar to our previous probabilistic transfer method (Chapter 7) with its fixed feature extractor after meta-training and an adaptive head model for the classifier; both build on top of similar probabilistic models for multi-task learning (Bakker and Heskes, 2003).

But in contrast to our previous approach, VERSA does not consider an explicit probabilistic prior model for the classifier weights. Instead, VERSA's objective

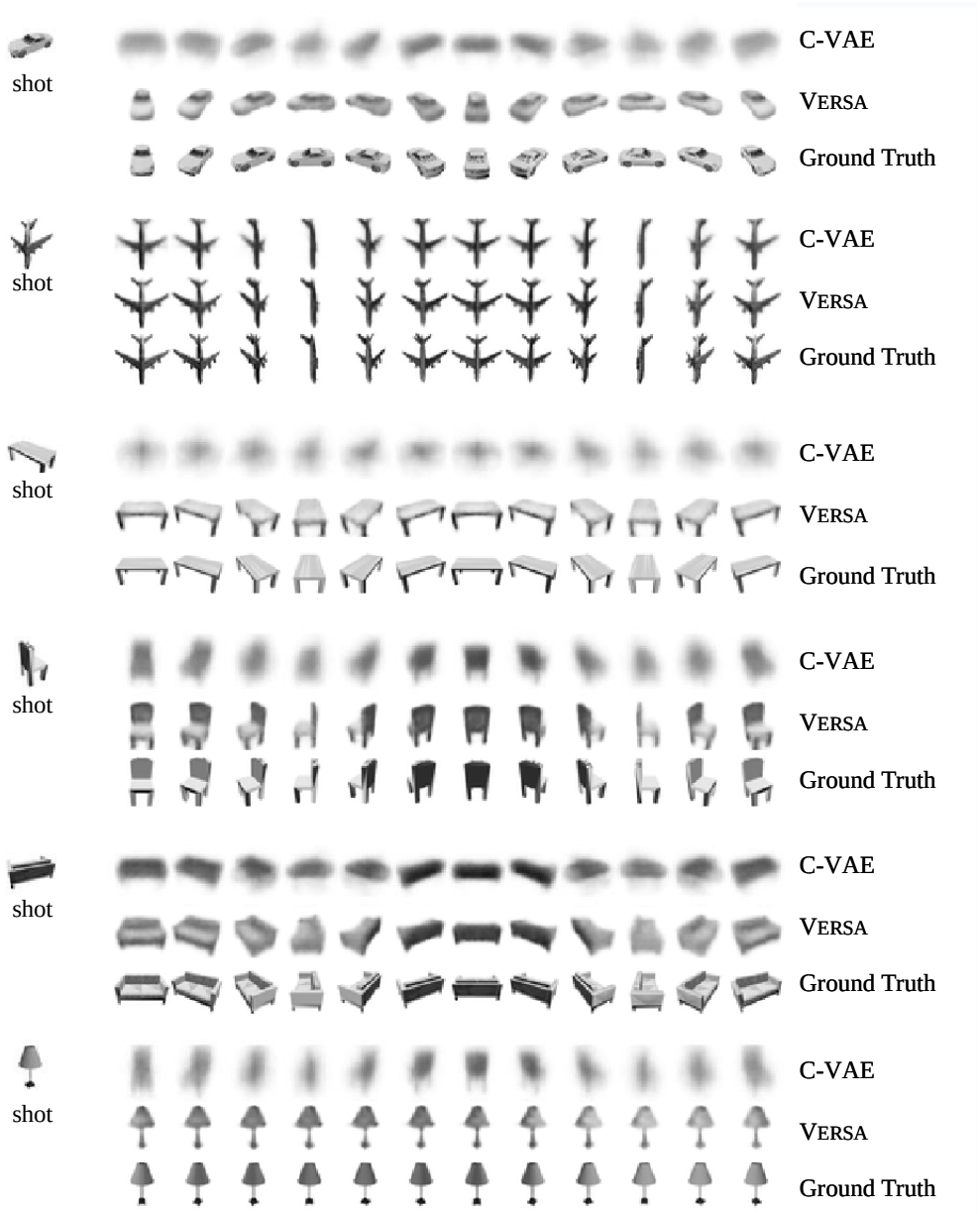


Figure 8.6: Results for ShapeNet view reconstruction for unseen objects from the test set (shown left). The model was trained to reconstruct views from a single orientation. *Top row:* images/views generated by a C-VAE model; *middle row* images/views generated by VERSA; *bottom row:* ground truth images. Views are spaced evenly every 30 degrees in azimuth.

function (Equation (8.15)) directly targets the posterior predictive distribution, in which the posterior over task-specific parameters is amortised through a recognition network on training sets. Therefore, we did not explicitly express the inductive biases in the probabilistic model and its prior, but rather implicitly encoded them in the training procedure and the recognition network, similar to the recently introduced conditional neural processes (Garnelo et al., 2018a). The recognition network acts as a meta-learner that adapts the distribution on the head model to new tasks.

In this chapter we embraced episodic multi-task learning and explicitly distinguish between training and testing data in the meta-train phase. While such a division into many small datasets appears unmotivated from the point of view of standard probabilistic inference, VERSA's objective and its episodic training procedure can be motivated through Bayesian decision theory (see Section 8.1.3). We saw in Table 8.1 that standard VI led to much poorer results compared to VERSA and speculated that this is due to over-regularisation by the explicit prior in that case.

As part of the model design, we employed a strong inductive bias by making the context independence assumption in both of our approaches. It allowed us to infer the head model weights separately for each class instead of all jointly. Not only does this reduce the number of parameters that need to be amortised or learned, it also makes the models more versatile and extensible to an arbitrary number of new classes. Yet, it would be interesting to explore ways to include at least some form of context in future work.

In terms of few-shot performance, our transfer approach outperforms VERSA and most other meta-learning approaches on 1- and 5-shot tasks on *mini*ImageNet: Our transfer approach obtains $56.3 \pm 0.4\%$ and $73.9 \pm 0.3\%$ respectively, whereas VERSA only obtains $53.4 \pm 1.82\%$ and $67.37 \pm 0.86\%$, respectively (compare Tables 7.4 and 8.1). This highlights the strength of transferable deep features that we discussed in Chapter 7 – the ResNet features are clearly more powerful than the features of more shallow CNNs introduced by Vinyals et al. (2016) and used in VERSA and most other meta-learning approaches. In practice, it is difficult to train such deep feature extractors in an episodic fashion, such that the literature focuses on few-shot performance with shallow CNN architectures to asses meta-learning abilities (also see our discussion above in Sections 7.3.7 and 8.4.2).

When powerful feature extractors exist for the data modality at hand, and pure few-shot performance on a narrow task is relevant, transfer approaches should therefore be preferred. VERSA on the other hand is more versatile and can be adapted to a wider range of settings as we illustrated with our view reconstruction

task. Moreover, we speculate that it performs better on tasks where no powerful feature extractors exist. We discuss this further in Chapter 9.

In Chapter 7 we have discussed some of the specific design choices of the feature extractor, such as its output dimensionality d_Φ. VERSA, in line with other meta-learning approaches for few-shot classification, also employs $d_\Phi = 256$ with a similar trade-off between expressiveness and complexity. Moreover, the division of work between the feature extractor and the head model varies between approaches, and we speculate that its optimal setting is also task dependent (see our discussion on deep features above). For both of our approaches the choice to treat only the last layer as task-dependent head was somewhat arbitrary and is mostly due to practical considerations, such as ease of modelling or the context independent approximation. In principle, we could for example employ multi-layer head models, and ML-PIP allows all of their parameters to be treated as task-dependent.

8.6 SUMMARY

This chapter introduced ML-PIP, a probabilistic framework for meta-learning. ML-PIP unifies a broad class of recently proposed meta-learning methods, and suggests alternative approaches.

Building on ML-PIP, we developed VERSA, a versatile few-shot learning algorithm that avoids the use of gradient based optimisation at test time by amortising posterior inference of task-specific parameters. We evaluated VERSA on several few-shot learning tasks and demonstrated state-of-the-art performance and compelling visual results on a challenging 1-shot view reconstruction task.

It would be interesting to assess the performance of VERSA or ML-PIP on other few-shot learning tasks, for example in regression beyond view reconstruction or reinforcement learning. Moreover, we assumed fixed image features after the initial meta-training phase similar to our previous approach in Chapter 7. Future work could investigate whether these image features can be adapted in an amortised fashion as well. The recently proposed residual adapters (Rebuffi et al., 2017) are a natural starting point for such an investigation, as they limit the number of adaptable parameters in the feature extractor.

EPILOGUE

9 DISCUSSION AND CONCLUSION

Learning is the ability to generalise beyond training examples. As discussed in the introduction, many generalisations may in principle be consistent with a given set of observations; inductive biases then help us to select certain generalisations over others. Mitchell (1980) lists different types of biases, for example limiting the expressiveness of models, integrating expert knowledge, or following heuristics such as Occam's razor. In this sense, all machine learning models rely on inductive biases to generalise.

In this thesis we focused on probabilistic machine learning, and asked more specifically, how different modelling choices and priors allow us to learn and make inferences from data for the tasks at hand. Because learning is rarely tractable in closed form, we discussed approximate inference through variational methods as a secondary theme.

Here, we first provide a brief overview of the main results of this thesis and then discuss unifying themes that relate to several chapters in this thesis. For more comprehensive summaries and discussions of the individual results, refer back to the respective chapters.

9.1 OVERVIEW OF THE MAIN RESULTS

First, we showed how general invariances can be integrated into Gaussian process priors and learned using the marginal likelihood to substantially improve performance (Chapter 4). The proposed inference scheme for large double-sum kernels can be applied beyond invariances to address intractable kernels in general. We also demonstrated that the involved intractabilities and computational constraints in GP methods are best addressed with variational approximations that behave predictably and actually yield a bound to the log marginal likelihood (Chapter 3).

We then presented a mechanism to define more flexible prior distributions using a form of rejection sampling (Chapter 5) to address the the mismatch between aggregate posteriors and priors in variational autoencoders. Our approach generalises to unseen test data and reduces the number of low quality samples from the generative model in a practical way.

Finally, we proposed two probabilistic approaches to few-shot learning from a transfer and meta-learning perspective, respectively, that both achieve state of the art results on benchmarks. The first focuses on simplicity and provides a strong baseline for few-shot learning . It relies on transferring deep features using a probabilistic model and can be linked to automatically regularised softmax regression (Chapter 7). The second emphasises flexibility and employs an amortised head model; it can be viewed as meta-learning probabilistic inference for prediction, and can be generalised to other contexts such as few-shot regression (Chapter 8). Its general framework called ML-PIP provides a unifying perspective on many recent approaches to few-shot learning.

9.2 INDUCTIVE BIASES IN PROBABILISTIC MODELLING

Here, we recapitulate the modelling steps of probabilistic machine learning and their inductive biases, and relate the results of this thesis to them. In the discussion we are guided by the following questions:

- *Where and how do we encode biases generally?*
- *Which inductive biases do priors encode and how do we encode particular biases?*
- *Which biases should we encode in the model and what should be learned from data?*

9.2.1 *Probabilistic modelling – Where and how to encode biases generally?*

When specifying a model, we typically write down relations between variables in terms of a graphical model that illustrates how the joint distribution factorises, and choose the distributions and their specific parametric or non-parametric realisation. These choices limit the relations that can be learned and functions that can be expressed in principle – even if we had access to an oracle that could choose the "best" configuration. By adding a prior on root factors, we further restrict this expressiveness and – at the same time – define the set of typical a priori configurations.

In summary, inductive biases are typically encoded in the model structure and priors; however, the inference scheme – especially when performing approximate inference – can have an influence as well. For example, variational inference tends to favour simpler solutions (Turner and Sahani, 2011).

9.2.2 *Which inductive biases do priors encode and how do we encode particular biases?*

The inductive biases of priors depend on the particular probabilistic model. GP priors describe the entire latent function, and expert biases can be incorporated

in its covariance function. By making the prior hyperparameters trainable, the data can inform the model about the best typical configurations out of a broader family of functions. In Chapter 4 we expanded the set of expert biases by general invariances, which can now be incorporated and learned using the marginal likelihood.

In contrast, the priors in BNNs are over individual weights, such that the induced inductive biases are much less interpretable – *What does a factorised Gaussian prior on weights actually mean?* Results by Neal (1994) but also Williams (1998) and Cho and Saul (2009) and more recently Matthews et al. (2018) provide a connection in terms of Gaussian Process priors for certain single and multi-layer networks and derive their covariance function. Regardless of this the inverse problem remains elusive – *How do we encode a particular (expert) bias in BNNs?*

From a modelling perspective it seems very sensible to place priors on functions rather than weights for regression tasks. An interesting recent approach by Sun et al. (2019) addresses the above inverse problem by placing function space constraints on parametric neural network models and approximating the involved KL. Similarly, both VERSA (Chapter 8) and neural processes (Garnelo et al., 2018a) model conditional densities and implicitly provide typical sets of functions rather than weights. Neither of them depend on explicit priors but encode the biases in their amortisation networks, instead. In fact, we showed that VERSA – with its ML-PIP objective that directly targets the posterior predictive distribution – outperforms a more traditional variational inference approach which approximates the log marginal likelihood (see Section 8.4.2), highlighting VI's tendency to underfit (Turner and Sahani, 2011).

Priors on the latent space of continuous latent variable models, and in particular VAEs, mostly serve as simple base distributions that are non-linearly transformed through the likelihood to match a data distribution. Implicit inductive biases are therefore encoded in the structure or architecture of the decoder or encoder with the prior merely encouraging smoothness and continuity in the latent space. At least, this is true for continuous latent spaces and the standard Normal prior. We can impose much stronger structure and inductive biases, for example by placing information constraints on the latent space (Zhao et al., 2019) or using mixed continuous and discrete latent variables (Vahdat et al., 2018).

The regularising effect of standard priors can lead to over-regularisation (see Section 1.6.4) that can be practically mitigated by making the prior more flexible; resampled LARS priors (Chapter 5) and similarly the Vamp prior (Tomczak and Welling, 2018) give rise to models with better performance and improved sample quality. So far, the role and effects of the prior in VAE training has been neglected compared to the approximate posterior and we speculate that research in this

direction could lead to further improvements, such as the very recent "energy inspired models" (Lawson et al., 2019).

9.2.3 *Which biases should be encoded and what should be learned from data?*

The question *How do we encode biases?* naturally leads to the question of *Which biases should we encode and what should be learned from data alone?* This question is a fundamental tension in machine learning; at one extreme, parameterised rule-based expert systems have very strong inductive biases and hardly require any data but are very specific. In contrast, systems such as Alpha Zero (Silver et al., 2018) or neural architecture searches (Elsken et al., 2019) compensate for the lack of rules or expert biases with compute power and ultimately through abundance of data and simulations (together with cleverly crafted inductive biases through their architectures). Most machine learning methods aim to be general but are subject to external constraints, such as data availability and compute power, which necessitate stronger biases. The trade-off and the choice of particular biases is problem-specific.

In Chapter 4 we proposed a GP covariance function that can encode general invariances; however, the strength of the incorporated biases is determined by the particular augmentation distribution which ultimately implicitly defines the set of invariances. We can therefore tune the trade-off between incorporated expert knowledge and learning from data by how we parameterise the augmentation distribution. We used very simple affine transformations and local deformations in Chapter 4 but nonetheless learn the degree to which we want to be invariant, which makes this approach superior to other data augmentation techniques that rely on cross-validation (Loosli et al., 2007). We speculated that more flexible augmentation distributions – which utilise generative models with many more learnable parameters – could be employed to uncover instance-dependent structures and invariances beyond these simple transformations.

Similarly, ML-PIP (Chapter 8) provides a general framework for probabilistic meta-learning with global and task-specific parameters; further biases are incorporated through the particular choice of instantiation, such as VERSA, which makes it adaptable to classification or regression tasks.

9.2.4 *Model complexity vs inference complexity*

Another trade-off that we have not explicitly discussed yet is between model complexity and inference complexity. Often, relatively complex models are paired with simple inference schemes, whereas simpler models can be paired with more

powerful inference methods. Often this is due to computational constraints when dealing with large models.

Our few-shot approaches in Chapters 7 and 8 are an example of this: Our transfer approach employs powerful deep features but a relatively simple transfer model for few-shot inference. In contrast, meta-learning approaches use very elaborate learning procedures but rely on relatively simple feature extractors.

Similarly, the inference scheme for invariant GPs (Chapter 4) is quite involved and we restrict ourselves to simple augmentation distributions, which are already slow to train. At the cost of computation time and optimisation difficulties we could replace them with more flexible transformations that can express a richer set of augmentations and invariances, as discussed above.

However, with more compute power and data becoming available, this trade-off is becoming less pronounced. For example, GANs are now trained with more complex inference procedures on top of relatively complex models. We therefore speculate, that probabilistic machine learning might see richer models with involved inference procedures to improve performance and ask: *Could VAEs, for example, perform better if we only engineered them better, similarly to what has been done for GANs?*

Finally, an important learning from Chapter 3 is that approximations and inference schemes should be principled and theoretically grounded rather than ad-hoc in order to allow for reliable diagnosis and avoid potential unwanted pathologies.

9.3 MODEL EVALUATION AND BENCHMARK DATASETS

After model training, an important step is model evaluation and comparison. The core of this empirical evaluation is often limited to a relatively small and slowly evolving set of "standard" benchmark datasets for each specific task accompanied by one or several baseline methods. For example, for standard image classification, MNIST and CIFAR-10/100 have been superseded by ImageNet.

Some important desiderata for good benchmarks are:

- realism – *How well do they represent the problem?*
- breadth – *Do they cover all different aspects of the problem?*
- discriminative power – *Can they distinguish between different methods? Are they not too hard and not too easy?*
- availability and easy of usage – *Can they be used easily?*

In addition, a thorough model evaluation should disentangle the contributions of each novel component, discuss potential alternatives and their performance, and compare against strong and canonical baselines. Canonical here refers to simple and straightforward methods with no or very few free parameters.

A conclusion we draw from our findings in Chapters 7 and 8 is that the presented few-shot image classification task or at least the currently employed datasets (Omniglot and *mini*ImageNet) are not necessarily the best benchmarks to assess transfer and meta-learning. Moreover, many meta-learning approaches do not compare to the strongest baselines.

In general for the discriminative few-shot learning task, it is difficult to disentangle which gains in performance are due to superior feature extractors and which are due to better concept transfer methods or meta-learners. For natural images there exist very strong feature extractors that favour transfer learning approaches, and in Chapter 7, we highlighted that the same concept transfer method can show very different results depending on the feature extractor used. In some cases, these variances were larger than differences between methods. Moreover, the quality of the feature extractors also heavily relies on the data augmentation strategies used, as they encourage particular inductive biases that generalise better to unseen classes as also discussed in Chapter 4. A potential quick fix could be a rigid standardisation of datasets, their augmentations, and feature extractors, at least when assessing the transfer or meta-learning ability of related approaches. Though, such limitations on feature extractors might not be entirely fair to transfer approaches – in Chapter 7 we highlighted that they can employ deep feature extractors whereas many meta-learning approaches are limited to much shallower architectures due to episodic training.

Despite these shortcomings, the few-shot learning community has focused almost exclusively on image classification tasks and only very recently included regression and reinforcement learning problems, which highlight other aspects of few-shot learning and rely less on good features.

Moreover, many recent works do not necessarily compare to strong baselines, such as our approach in Chapter 7, which can be linked to L2 regularised softmax regression with pre-trained deep features and provides a natural and canonical baseline. Very recently, Chen et al. (2019) proposed a related approach in a comparative study, which further highlights the importance of strong baselines.

We now address individual image few-shot learning datasets in particular. Omniglot – similarly to MNIST for regular classification – appears to be a very simple problem with most methods achieving more than 97% accuracy in few-shot tasks, see Chapter 8. Due to this relative simplicity and small size, it can be used

for fast prototyping and sanity checks. However, it only very weakly discriminates between "strong" and "weak" methods.

*mini*ImageNet is substantially harder but is still relatively small and the distribution of classes is imbalanced. Moreover, the split into meta-train and meta-test classes is random, such that similar classes could be shared between them. Consequently, the features learned in the pre-training or meta-training phase could already be highly discriminative for some of the test classes and skew performance. Very recently, Ren et al. (2018) proposed *tiered*ImageNet, which also partitions ImageNet classes more systematically according to the root nodes of the associated WordNet tree. In the same vein, Triantafillou et al. (2018) introduced 'Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples'. However, the machine learning community has yet to agree on a systematic set of benchmarks that highlight strengths and weaknesses of different few-shot learning approaches and do not suffer from the above artifacts.

In VAEs two types of benchmarks are typically used to evaluate models: relatively simple grey scale images – MNIST, Omniglot, and FashionMNIST – and more complicated colour images – typically CIFAR-10/100 or downscaled versions of ImageNet. Many approaches, including the one in Chapter 5, are only evaluated on the grey scale benchmarks due mostly to computation constraints. However, it is unclear how well the reported results translate to more challenging problems.

Similarly to the few-shot learning task, the benchmarks again focus on image datasets; however, many challenging questions arise when addressing less structured data, where even simple tasks such as imputation prove difficult (Nazábal et al., 2018). In many cases, these problems are more challenging and in some sense more realistic than several of the image benchmarks.

We have not extensively discussed benchmarks for GP models throughout this thesis. Probabilistic methods for regression such as GPs, DeepGPs (Damianou and Lawrence, 2013), and BNNs are often evaluated on several simple UCI regression datasets. While they provide reasonable discrimination and are of different sizes, it is unclear how realistic or representative they are, too.

9.4 ADVICE FOR PRACTITIONERS

Results in this thesis also provide direct advice for practitioners. In general, while complex and general approaches often perform well on benchmarks and provide flexibility, they are usually more difficult to set up and harder to diagnose and

interpret than simpler approaches; this is especially true when the full generality and flexibility of more advanced approaches is not necessary.

For example, when addressing many small but related tasks in few-shot learning in Chapters 7 and 8, we found that our transfer approach that uses a combination of a pre-trained feature extractor and a simple probabilistic head model often works surprisingly well compared to many more complex meta-learning models. It also provided good calibration, and is easy to use and interpret in practice, see Chapter 7. These results are in line with earlier research that learned deep features are often surprisingly transferable to new discriminative tasks, though they may require some fine-tuning. In other words, when good feature extractors and a large enough training repository are available for a given data modality (as is the case for images but also text and audio), an approach which reuses these features in a sensibly regularised way constitutes at least a strong baseline. In Chapter 7 we exemplified this for the case for natural images and our probabilistic approach allowed us to automatically choose the regularisation in a principled way. In practical applications such an approach might be both faster and yield solid and more robust results compared to more complicated approaches. While optimisation at test-time is necessary, it is also relatively fast due to the simplicity of the model.

More recently, Chen et al. (2019) show empirically, that deeper feature extractors applied consistently to many approaches actually reduce performance differences between many methods. They propose a simple baseline method very similar to our transfer approach in Chapter 7 with very similar results in terms of competitive performance.

In our opinion, more complex meta-learning approaches should only be used when more flexibility is required or transfer approaches do not provide good or fast enough results, for example when no good pre-trainable feature extractors exist for a particular data modality or domain.

For smaller to medium sized regression problems, Gaussian process methods are still a method of choice, especially when good predictive uncertainties are required, such as for active learning or Bayesian optimisation. To make these methods more scalable, practitioners can choose from a wide range of approximations. We concluded from Chapter 3 that variational methods (Titsias, 2009a; Hensman et al., 2013, 2015) should be preferred over less principled and more ad-hoc approximations such as FITC (Snelson and Ghahramani, 2006) or DTC (Seeger et al., 2003) due to pathologies of their objective function especially for FITC. While Turner and Sahani (2011) discuss several problems of variational methods such as underfitting, our results suggest that they can often be mediated

and, crucially, both the bound as well as the noise variance can still be used as informative diagnostics for regression.

Especially for simpler VAEs, samples can vary hugely in quality due to the mismatch between the aggregate posterior and the prior. While Rosca et al. (2018) provide tools to quantify this mismatch, the more flexible LARS or resampled priors presented in Chapter 5 can be used to automatically reject samples of poor quality as they mostly correspond to samples from the mismatched regions; LARS priors can be trained post-hoc to already trained VAEs to fulfil this role.

10

BIBLIOGRAPHY

Mauricio Álvarez, David Luengo, Michalis Titsias, and Neil D. Lawrence (2010). 'Efficient Multioutput Gaussian Processes through Variational Inducing Kernels'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (page 40).

Shun-ichi Amari (2016). *Information Geometry and Its Applications*. 1st. Springer Publishing Company, Incorporated (page 14).

Antreas Antoniou, Amos Storkey, and Harrison Edwards (2017). 'Data Augmentation Generative Adversarial Networks'. *arXiv preprint arXiv:1711.04340* (page 71).

Bart Bakker and Tom Heskes (2003). 'Task Clustering and Gating for Bayesian Multitask Learning'. *Journal of Machine Learning Research* (pages 137, 140, 143, 144, 167, 175, 186).

Matthias Bauer and Andriy Mnih (2019). 'Resampled Priors for Variational Autoencoders'. In: *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics* (pages 4, 6, 97).

Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen (2016). 'Understanding Probabilistic Sparse Gaussian Process Approximations'. In: *Advances in Neural Information Processing Systems 29* (pages 3, 5, 51).

Matthias Bauer, Valentin Volchkov, Michael Hirsch, and Bernhard Schölkopf (2018). 'Automatic estimation of modulation transfer functions'. In: *2018 IEEE International Conference on Computational Photography (ICCP)* (page 5).

Matthias Bauer*, Johannes Knebel*, Matthias Lechner, Peter Pickl, and Erwin Frey (2017a). 'Ecological feedback in quorum-sensing microbial populations can induce heterogeneous production of autoinducers'. *eLife* (page 6).

Matthias Bauer*, Mateo Rojas-Carulla*, Jakub B. Świątkowski, Bernhard Schölkopf, and Richard E. Turner (2017b). 'Discriminative k-shot learning using probabilistic models'. In: *Second Workshop on Bayesian Deep Learning at the 31st Conference on Neural Information Processing Systems* (pages 4, 6, 143, 175).

James O. Berger (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media (page 172).

David Beymer and Tomaso Poggio (1995). 'Face recognition from one example view'. In: *Proceedings of IEEE International Conference on Computer Vision* (page 71).

Christopher M. Bishop (2006). *Pattern Recognition and Machine Learning*. Springer Verlag New York Inc. (pages 12, 24).

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe (2017). 'Variational Inference: A Review for Statisticians'. *Journal of the American Statistical Association* (page 14).

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). 'Weight Uncertainty in Neural Network'. In: *International Conference on Machine Learning* (page 14).

Edwin V Bonilla, Daniel Steinberg, and Alistair Reid (2016). 'Extended and Unscented Kitchen Sinks'. In: *Proceedings of The 33rd International Conference on Machine Learning* (page 49).

Aleksandar Botev, Bowen Zheng, and David Barber (2017). 'Complementary Sum Sampling for Likelihood Approximation in Large Scale Classification'. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (pages 108, 109).

Olivier Bousquet and Daniel J. L. Herrmann (2003). 'On the Complexity of Learning the Kernel Matrix'. In: *Advances in Neural Information Processing Systems 15*. MIT Press (page 19).

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio (2016). 'Generating Sentences from a Continuous Space'. In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning* (pages 33, 116).

Thang D. Bui, Josiah Yan, and Richard E. Turner (2016). 'A Unifying Framework for Sparse Gaussian Process Approximation using Power Expectation Propagation' (page 43).

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov (2016). 'Importance weighted autoencoders'. In: *Proceedings of the 4th International Conference on Learning Representations* (pages 31, 33, 117).

Jordan Burgess, James Robert Lloyd, and Zoubin Ghahramani (2016). 'One-Shot Learning in Discriminative Neural Networks'. *NIPS Bayesian Deep Learning workshop* (pages 141, 143).

David Burt, Carl Edward Rasmussen, and Mark van der Wilk (2019). 'Rates of Convergence for Sparse Variational Gaussian Process Regression'. In: *Proceedings of the 36th International Conference on Machine Learning* (page 67).

Rich Caruana (1997). 'Multitask Learning'. PhD thesis. Carnegie Mellon University (page 135).

George Casella and Roger L Berger (2002). *Statistical inference*. Duxbury Pacific Grove, CA (page 171).

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu (2015). *ShapeNet: An Information-Rich 3D Model Repository*. Technical report. Stanford University — Princeton University — Toyota Technological Institute at Chicago (pages 185, 238).

Olivier Chapelle and Bernhard Schölkopf (2002). 'Incorporating Invariances in Non-Linear Support Vector Machines'. In: *Advances in Neural Information Processing Systems 14* (page 72).

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang (2019). 'A Closer Look at Few-shot Classification'. In: *International Conference on Learning Representations* (pages 138, 198, 200).

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel (2017). 'Variational Lossy Autoencoder'. In: *Proceedings of the 5th International Conference on Learning Representations* (pages 33, 100, 117, 128).

Youngmin Cho and Lawrence K. Saul (2009). 'Kernel Methods for Deep Learning'. In: *Advances in Neural Information Processing Systems 22* (page 195).

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter (2016). 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)'. In: *4th International Conference on Learning Representations (ICLR)* (page 232).

Taco Cohen and Max Welling (2016). 'Group Equivariant Convolutional Networks'. In: *Proceedings of the 33rd International Conference on Machine Learning* (page 72).

Chris Cremer, Xuechen Li, and David Duvenaud (2018). 'Inference Suboptimality in Variational Autoencoders'. In: *Proceedings of the 35th International Conference on Machine Learning* (pages 14, 15, 183).

Lehel Csató and Manfred Opper (2001). 'Sparse representation for Gaussian process models'. In: *Advances in Neural Information Processing Systems* (page 42).

— (2002). 'Sparse on-line Gaussian processes'. *Neural computation* (pages 40–42).

John P. Cunningham, Krishna V. Shenoy, and Maneesh Sahani (2008). 'Fast Gaussian Process Methods for Point Process Intensity Estimation'. In: *Proceedings of the 25th International Conference on Machine Learning* (page 49).

George Cybenko (1989). 'Approximation by superpositions of a sigmoidal function'. *Mathematics of Control, Signals and Systems* (page 1).

Andreas C. Damianou and Neil D. Lawrence (2013). 'Deep Gaussian processes'. In: *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics* (page 199).

Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Re (2019). 'A Kernel Theory of Modern Data Augmentation'. In: *Proceedings of the 36th International Conference on Machine Learning* (page 77).

A Philip Dawid (2007). 'The geometry of proper scoring rules'. *Annals of the Institute of Statistical Mathematics* (page 173).

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). 'ImageNet: A Large-Scale Hierarchical Image Database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (page 71).

Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous (2017). 'TensorFlow Distributions'. *CoRR* (page 229).

Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan (2016). 'Deep unsupervised clustering with gaussian mixture variational autoencoders'. *arXiv preprint arXiv:1611.02648* (page 128).

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio (2017). 'Density estimation using Real NVP'. In: *Proceedings of the 5th International Conference on Learning Representations* (pages 30, 110).

Arnaud Doucet, Simon Godsill, and Christophe Andrieu (2000). 'On Sequential Monte Carlo Sampling Methods for Bayesian Filtering'. *Statistics and Computing* (page 14).

David Duvenaud (2014). 'Automatic Model Construction with Gaussian Processes'. PhD thesis. University of Cambridge (page 18).

Harrison Edwards and Amos Storkey (2017). 'Towards a neural statistician'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (pages 133, 170, 184).

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter (2019). 'Neural Architecture Search: A Survey'. *Journal of Machine Learning Research* (page 196).

Li Fei-Fei, Rob Fergus, and Pietro Perona (2006). 'One-shot learning of object categories'. *IEEE transactions on pattern analysis and machine intelligence* (page 133).

Anibal Figueiras-Vidal and Miguel Lázaro-Gredilla (2009). 'Inter-domain Gaussian processes for sparse inference using inducing features'. In: *Advances in Neural Information Processing Systems 22* (page 80).

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih (2018). 'Implicit Reparameterization Gradients'. In: *Advances in Neural Information Processing Systems 31* (page 30).

Michael Fink (2005). 'Object Classification from a Single Example Utilizing Class Relevance Metrics'. In: *Advances in Neural Information Processing Systems 17* (page 133).

Chelsea Finn, Pieter Abbeel, and Sergey Levine (2017). 'Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks'. In: *Proceedings of the 34th International Conference on Machine Learning* (pages 133, 140, 160, 161, 169, 178, 183, 184).

Chelsea Finn, Kelvin Xu, and Sergey Levine (2018). 'Probabilistic Model-Agnostic Meta-Learning'. In: *Advances in Neural Information Processing Systems 31* (pages 140, 184).

Robert M. French (1999). 'Catastrophic forgetting in connectionist networks'. *Trends in cognitive sciences* (page 165).

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami (2018a). 'Conditional Neural

Processes'. In: *Proceedings of the 35th International Conference on Machine Learning* (pages 138, 178, 180, 188, 195).

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh (2018b). 'Neural processes'. *arXiv preprint arXiv:1807.01622* (page 180).

Pascal Germain, Francis Bach, Alexandre Lacoste, and Simon Lacoste-Julien (2016). 'PAC-Bayesian Theory Meets Bayesian Inference'. In: *Advances in Neural Information Processing Systems 29* (page 13).

Zoubin Ghahramani (2013). 'Bayesian non-parametrics and the probabilistic approach to modelling'. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* (pages 9, 12).

— (2015). 'Probabilistic machine learning and artificial intelligence'. *Nature* (pages 1, 13).

Mark N. Gibbs and David J. C. MacKay (2000). 'Variational Gaussian process classifiers'. *IEEE Transactions on Neural Networks* (page 85).

Spyros Gidaris and Nikos Komodakis (2018). 'Dynamic Few-Shot Visual Learning without Forgetting'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (page 182).

David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro (2012). 'Argumentwise invariant kernels for the approximation of invariant functions'. *Annales de la Faculté de Sciences de Toulouse* (pages 72, 73, 76).

David Ginsbourger, Olivier Roustant, and Nicolas Durrande (2013). 'Invariances of random fields paths, with applications in Gaussian Process Regression'. *arXiv preprint arXiv:1308.1359* (page 72).

— (2016). 'On degeneracy and invariances of random fields paths with applications in Gaussian process modelling'. *Journal of Statistical Planning and Inference* (page 72).

Xavier Glorot and Yoshua Bengio (2010). 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (page 116).

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik (2018). 'Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules'. *ACS Central Science* (page 25).

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). 'Generative Adversarial Nets'. In: *Advances in Neural Information Processing Systems 27* (page 25).

Jonathan Gordon*, John Bronskill*, Matthias Bauer, Sebastian Nowozin, and Richard Turner (2019). 'Meta-Learning Probabilistic Inference for Prediction'. In: *Proceedings of the 7th International Conference on Learning Representations* (pages 5, 6, 168).

Jonathan Gordon*, John Bronskill*, Matthias Bauer*, Sebastian Nowozin, and Richard E. Turner (2018a). 'Consolidating the Meta-Learning Zoo: A Unifying Perspective as Posterior Predictive Inference'. In: *Workshop on Meta-Learning (MetaLearn 2018) at the 32nd Conference on Neural Information Processing Systems* (pages 6, 168).

— (2018b). 'Versa: Versatile and Efficient Few-shot Learning'. In: *Third Workshop on Bayesian Deep Learning at the 32nd Conference on Neural Information Processing Systems* (pages 6, 168).

Thore Graepel and Ralf Herbrich (2004). 'Invariant Pattern Recognition by Semi-Definite Programming Machines'. In: *Advances in Neural Information Processing Systems 16* (page 72).

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths (2018). 'Recasting Gradient-Based Meta-Learning as Hierarchical Bayes'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (pages 167, 169, 178, 184).

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra (2015). 'DRAW: A Recurrent Neural Network For Image Generation'. In: *Proceedings of the 32nd International Conference on Machine Learning* (page 128).

Thomas L. Griffiths and Zoubin Ghahramani (2011). 'The Indian Buffet Process: An Introduction and Review'. *Journal of Machine Learning Research* (page 151).

Aditya Grover, Ramki Gummadi, Miguel Lazaro-Gredilla, Dale Schuurmans, and Stefano Ermon (2018). 'Variational Rejection Sampling'. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics* (page 129).

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville (2017). 'PixelVAE: A Latent Variable Model for Natural Images'. In: *Proceedings of the 5th International Conference on Learning Representations* (pages 100, 128).

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger (2017). 'On Calibration of Modern Neural Networks'. In: *Proceedings of the 34th International Conference on Machine Learning* (pages 155, 163, 164).

David Ha, Andrew Dai, and Quoc V. Le (2017). 'HyperNetworks'. In: *Proceedings of the 5th International Conference on Learning Representations* (page 179).

Frederik Harder, Matthias Bauer, and Mijung Park (2020). 'Interpretable and Differentially Private Predictions'. In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence* (page 6).

Bharath Hariharan and Ross Girshick (2017). 'Low-Shot Visual Recognition by Shrinking and Hallucinating Features'. In: *The IEEE International Conference on Computer Vision (ICCV)* (page 139).

Søren Hauberg (2018). 'Only Bayes should learn a manifold' (page 24).

Søren Hauberg, Oren Freifeld, Anders Boesen Lindbo Larsen, John Fisher, and Lars Hansen (2016). 'Dreaming More Data: Class-dependent Distributions over Diffeomorphisms for Learned Data Augmentation'. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (page 71).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (pages 71, 136, 160, 231).

James Hensman, Nicolo Fusi, and Neil D. Lawrence (2013). 'Gaussian Processes for Big Data'. In: *Conference on Uncertainty in Artificial Intelligence* (pages 46, 47, 58, 66, 67, 79, 200).

James Hensman, Alexander G. de G. Matthews, and Zoubin Ghahramani (2015). 'Scalable Variational Gaussian Process Classification'. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics* (pages 47, 48, 66, 67, 80, 200).

Tom Heskes (2000). 'Empirical Bayes for Learning to Learn'. In: *Proceedings of the Seventeenth International Conference on Machine Learning* (pages 137, 167, 175).

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). 'β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework'. In: *Proceedings of the 5th International Conference on Learning Representations* (pages 24, 121).

Geoffrey Hinton (2000). 'Training Products of Experts by Minimizing Contrastive Divergence'. *Neural Computation* (page 129).

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal (1995). 'The" wake-sleep" algorithm for unsupervised neural networks'. *Science* (pages 29, 173).

Sepp Hochreiter and Jürgen Schmidhuber (1997). 'Long Short-Term Memory'. *Neural Computation* (page 140).

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley (2013). 'Stochastic Variational Inference'. *The Journal of Machine Learning Research* (page 46).

Matthew D. Hoffman and Andrew Gelman (2014). 'The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo.' *Journal of Machine Learning Research* (pages 154, 157).

Matthew D. Hoffman and Matthew J. Johnson (2016). 'ELBO Surgery'. *NIPS 2016 Workshop on Advances in Approximate Bayesian Inference* (pages 32, 33, 97, 98, 122).

Kurt Hornik, Maxwell Stinchcombe, and Halbert White (1989). 'Multilayer feedforward networks are universal approximators'. *Neural Networks* (page 1).

H. Hotelling (1933). 'Analysis of a complex of statistical variables into principal components.' *Journal of Educational Psychology* (page 24).

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville (2018). 'Neural Autoregressive Flows'. In: *Proceedings of the 35th International Conference on Machine Learning* (page 111).

Ferenc Huszar (2013). 'Scoring rules, divergences and information in Bayesian machine learning'. PhD thesis. University of Cambridge (page 173).

Sergey Ioffe and Christian Szegedy (2015). 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'. In: *International Conference on Machine Learning* (page 232).

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros (2017). 'Image-to-Image Translation with Conditional Adversarial Networks'. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (page 92).

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu (2015). 'Spatial Transformer Networks'. In: *Advances in Neural Information Processing Systems 28* (pages 72, 87, 92).

Edwin T. Jaynes (2003). *Probability Theory: The Logic of Science*. Cambridge University Press (page 172).

William H. Jefferys and James O. Berger (1992). 'Ockham's Razor and Bayesian Analysis'. *American Scientist* (page 13).

Ian T. Jolliffe (1986). *Principal Component Analysis*. Springer (page 24).

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul (1999). 'An Introduction to Variational Methods for Graphical Models'. *Machine Learning* (page 14).

Łukasz Kaiser, Ofir Nachum, Roy Aurko, and Samy Bengio (2017). 'Learning to Remember Rare Events'. In: *Proceedings of the 5th International Conference on Learning Representations* (page 184).

Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker (1988). 'Design of an image edge detection filter using the Sobel operator'. *IEEE Journal of solid-state circuits* (page 88).

Yoon Kim, Sam Wiseman, Andrew Miller, David Sontag, and Alexander Rush (2018). 'Semi-Amortized Variational Autoencoders'. In: *Proceedings of the 35th International Conference on Machine Learning* (pages 30, 178).

Diederik P. Kingma and Jimmy Ba (2015). 'Adam: A Method for Stochastic Optimization'. In: *Proceedings of the 3rd International Conference on Learning Representations* (pages 116, 181, 232, 236, 238).

Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling (2014). 'Semi-supervised learning with deep generative models'. In: *Advances in Neural Information Processing Systems* (page 186).

Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling (2016). 'Improved Variational Inference with Inverse Autoregressive Flow'. In: *Advances in Neural Information Processing Systems 29* (pages 30, 117).

Diederik P Kingma, Tim Salimans, and Max Welling (2015). 'Variational Dropout and the Local Reparameterization Trick'. In: *Advances in Neural Information Processing Systems 28* (pages 174, 237).

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov (2015). 'Siamese neural networks for one-shot image recognition'. In: *ICML Deep Learning Workshop* (pages 138, 184).

Risi Kondor (2008). 'Group theoretical methods in machine learning'. PhD thesis. Columbia University (pages 72, 73, 76).

Alex Krizhevsky (2009). 'Learning multiple layers of features from tiny images' (page 155).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (2012). 'Imagenet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems 25* (page 71).

Dirk P. Kroese, Thomas Taimre, and Zdravko I. Botev (2013). *Handbook of Monte Carlo Methods*. Wiley (page 107).

Simon Lacoste–Julien, Ferenc Huszár, and Zoubin Ghahramani (2011). 'Approximate inference for the loss-calibrated Bayesian'. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (page 173).

Alexandre Lacoste, Boris Oreshkin, Wonchang Chung, Thomas Boquet, Negar Rostamzadeh, and David Krueger (2018). 'Uncertainty in Multitask Transfer Learning'. *arXiv preprint arXiv:1806.07528* (pages 133, 182).

Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum (2011). 'One shot learning of simple visual concepts'. In: *Proceedings of the Annual Meeting of the Cognitive Science Society* (pages 133, 182, 236).

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum (2015). 'Human-level concept learning through probabilistic program induction'. *Science* (page 115).

Hugo Larochelle and Iain Murray (2011). 'The Neural Autoregressive Distribution Estimator'. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (page 115).

Neil D. Lawrence (2004). 'Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data'. In: *Advances in Neural Information Processing Systems 16* (page 23).

Neil D. Lawrence, John C. Platt, and Michael I. Jordan (2005). 'Extensions of the Informative Vector Machine'. In: *Deterministic and Statistical Methods in Machine Learning* (page 71).

Neil D. Lawrence, Matthias Seeger, and Ralf Herbrich (2003). 'Fast Sparse Gaussian Process Methods: The Informative Vector Machine'. In: *Advances in Neural Information Processing Systems 15* (page 37).

John Lawson, George Tucker, Bo Dai, and Rajesh Ranganath (2019). 'Energy-Inspired Models: Learning with Sampler-Induced Distributions'. In: *Advances in Neural Information Processing Systems 32* (pages 101, 128, 196).

Miguel Lázaro-Gredilla and Anibal Figueiras-Vidal (2009). 'Inter-domain Gaussian processes for sparse inference using inducing features'. In: *Advances in Neural Information Processing Systems* (pages 40, 66).

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal (2010). 'Sparse Spectrum Gaussian Process Regression'. *Journal of Machine Learning Research* (page 48).

Quoc Le, Tamás Sarlós, and Alexander Smola (2013). 'Fastfood - approximating kernel expansions in loglinear time'. In: *Proceedings of the international conference on machine learning* (page 48).

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). 'Backpropagation Applied to Handwritten Zip Code Recognition'. *Neural Computation* (page 71).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). 'Gradient-based learning applied to document recognition'. *Proceedings of the IEEE* (pages 71, 115).

Yann Lecun, Sumit Chopra, Raia Hadsell, Marc Aurelio Ranzato, and Fu Jie Huang (2006). 'A tutorial on energy-based learning'. English (US). In: *Predicting structured data* (page 101).

Christiane Lemke, Marcin Budka, and Bogdan Gabrys (2015). 'Metalearning: a survey of trends and technologies'. *Artificial Intelligence Review* (page 134).

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li (2017). 'Meta-sgd: Learning to learn quickly for few shot learning'. *arXiv preprint arXiv:1707.09835* (page 184).

Scott Linderman, Matthew Johnson, and Ryan P Adams (2015). 'Dependent multinomial models made easy: Stick-breaking with the Pòlya-Gamma augmentation'. In: *Advances in Neural Information Processing Systems 28* (page 85).

Dong C. Liu and Jorge Nocedal (1989). 'On the limited memory BFGS method for large scale optimization'. *Mathematical programming* (pages 21, 154).

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem (2019). 'Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations'. In: *Proceedings of the 36th International Conference on Machine Learning* (page 25).

Gaëlle Loosli, Stéphane Canu, and Léon Bottou (2007). 'Training invariant support vector machines using selective sampling'. *Large scale kernel machines* (pages 71, 86, 87, 93, 196).

Xiaoyu Lu, Javier Gonzalez, Zhenwen Dai, and Neil Lawrence (2018). 'Structured Variationally Auto-encoded Optimization'. In: *Proceedings of the 35th International Conference on Machine Learning* (pages 19, 25).

Laurens van der Maaten and Geoffrey Hinton (2008). 'Visualizing data using t-SNE'. *Journal of Machine Learning Research* (pages 160, 234, 235).

David J. C. MacKay (1992). 'Bayesian Interpolation'. *Neural Computation* (page 51).

— (1995). 'Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks'. *Network: Computation in Neural Systems* (page 9).

— (1998). 'Introduction to Gaussian Processes'. *NATO ASI Series F Computer and Systems Sciences* (pages 18, 72).

— (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press (pages 7, 9, 12–14, 32).

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh (2017). 'The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables'. In: *International Conference on Learning Representations* (page 30).

James Martens (2014). 'New insights and perspectives on the natural gradient method' (page 14).

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* (pages 21, 231).

Emile Mathieu, Tom Rainforth, N Siddharth, and Yee Whye Teh (2019). 'Disentangling Disentanglement in Variational Autoencoders'. In: *Proceedings of the 36th International Conference on Machine Learning* (page 25).

Alexander G. de G. Matthews (2016). 'Scalable Gaussian process inference using variational methods'. PhD thesis. University of Cambridge (pages 56, 63).

Alexander G. de G. Matthews, James Hensman, Richard E. Turner, and Zoubin Ghahramani (2016). 'On Sparse variational methods and the Kullback-Leibler divergence between stochastic processes'. In: *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Statistics* (pages 46, 47, 80).

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani (2018). 'Gaussian Process Behaviour in Wide Deep Neural Networks'. In: *International Conference on Learning Representations* (page 195).

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman (2017). 'GPflow: A Gaussian process library using TensorFlow'. *Journal of Machine Learning Research* (page 21).

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger (2017). 'Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks'. In: *Proceedings of the 34th International Conference on Machine Learning* (page 129).

Thomas P. Minka (2001). 'Expectation propagation for approximate Bayesian inference'. In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. (pages 14, 42).

Thomas P. Minka and Rosalind W. Picard (1997). *Learning How to Learn is Learning with Point Sets* (page 134).

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel (2018). 'A Simple Neural Attentive Meta-Learner'. In: *International Conference on Learning Representations* (page 184).

Tom M. Mitchell (1980). *The Need for Biases in Learning Generalizations*. Technical report (pages 1, 9, 193).

Shakir Mohamed (2018). *Density Ratio Trick*. http://blog.shakirm.com/2018/01/machine-learning-trick-of-the-day-7-density-ratio-trick/. Blog (pages 176, 233).

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih (2019). 'Monte Carlo Gradient Estimation in Machine Learning' (page 30).

Kevin Murphy (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press (pages 9, 10, 150, 151).

Iain Murray, David MacKay, and Ryan P Adams (2009). 'The Gaussian Process Density Sampler'. In: *Advances in Neural Information Processing Systems 21* (pages 100, 101, 104, 128, 129).

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht (2015). 'Obtaining Well Calibrated Probabilities Using Bayesian Binning'. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (page 155).

Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei (2017). 'Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms'. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (page 129).

Devang K. Naik and RJ Mammone (1992). 'Meta-neural networks that learn by learning'. In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. IEEE (pages 134, 167).

Eric Nalisnick, Lars Hertel, and Padhraic Smyth (2016). 'Approximate inference for deep latent gaussian mixtures'. In: *NIPS Workshop on Bayesian Deep Learning* (page 128).

Siddharth Narayanaswamy, T Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr (2017). 'Learning Disentangled Representations with Semi-Supervised Deep Generative Models'. In: *Advances in Neural Information Processing Systems* (page 186).

Alfredo Nazábal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera (2018). 'Handling Incomplete Heterogeneous Data using VAEs'. *CoRR* (page 199).

Radford M. Neal (1993). *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Technical report (page 13).

— (1994). 'Bayesian Learning for Neural Networks'. PhD thesis. Dept. of Computer Science, University of Toronto (pages 9, 10, 51, 195).

— (2011). *MCMC using Hamiltonian dynamics* (pages 13, 154).

Andrew Y. Ng and Michael I. Jordan (2002). 'On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes'. In: *Advances in Neural Information Processing Systems* (page 168).

Alex Nichol and John Schulman (2018). 'Reptile: a Scalable Metalearning Algorithm'. *arXiv preprint arXiv:1803.02999* (pages 140, 184).

Partha Niyogi, Federico Girosi, and Tomaso Poggio (1998). 'Incorporating prior information in machine learning by creating virtual examples'. *Proceedings of the IEEE* (page 71).

Emmy Noether (1918). 'Invariante Variationsprobleme'. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* (page 70).

Boris Oreshkin, Pau Rodriguez López, and Alexandre Lacoste (2018). 'TADAM: Task dependent adaptive metric for improved few-shot learning'. In: *Advances in Neural Information Processing Systems 31* (page 182).

Diederik Kingma P. and Max Welling (2014). 'Auto-Encoding Variational Bayes'. In: *Proceedings of the 2nd International Conference on Learning Representations* (pages 15, 23, 25, 28, 29, 38, 85, 97, 167, 170).

Sinno J. Pan and Qiang Yang (2010). 'A Survey on Transfer Learning'. *IEEE Trans. on Knowl. and Data Eng.* (Page 134).

George Papamakarios, Iain Murray, and Theo Pavlakou (2017). 'Masked Autoregressive Flow for Density Estimation'. In: *Advances in Neural Information Processing Systems 30* (pages 30, 119, 128, 229).

K. Pearson (1901). 'On Lines and Planes of Closest Fit to Systems of Points in Space'. *Philosophical Magazine* (6) (page 24).

Kaare B. Petersen and Michael S. Pedersen (2012). *The Matrix Cookbook* (pages 21, 39).

Nicholas G. Polson, James G. Scott, and Jesse Windle (2013). 'Bayesian Inference for Logistic Models Using Pòlya–Gamma Latent Variables'. *Journal of the American Statistical Association* (page 85).

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas (2017). 'Pointnet: Deep learning on point sets for 3d classification and segmentation'. In: *The IEEE Conference on Computer Vision and Pattern Recognition* (page 175).

Yuan Qi, Ahmed H. Abdel-Gawad, and Thomas P. Minka (2010). 'Sparse-posterior Gaussian Processes for general likelihoods'. In: *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (page 43).

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L. Yuille (2018). 'Few-Shot Image Recognition by Predicting Parameters from Activations'. In: *The IEEE Conference on Computer Vision and Pattern Recognition* (pages 141, 175, 179, 182).

Joaquin Quiñonero-Candela and Carl Edward Rasmussen (2005). 'A unifying view of sparse approximate Gaussian process regression'. *The Journal of Machine Learning Research* (pages 16, 37, 38, 40, 41).

Sébastien Quirion, Chantale Duchesne, Denis Laurendeau, and Mario Marchand (2008). 'Comparing gplvm approaches for dimensionality reduction in character animation'. *J. WSCG* (page 24).

Ali Rahimi and Benjamin Recht (2008). 'Random Features for Large-Scale Kernel Machines'. In: *Advances in Neural Information Processing Systems 20* (page 48).

— (2009). 'Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning'. In: *Advances in Neural Information Processing Systems* (page 48).

Tom Rainforth, Adam Kosiorek, Tuan Anh Le, Chris Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh (2018). 'Tighter Variational Bounds are Not Necessarily Better'. In: *Proceedings of the 35th International Conference on Machine Learning* (page 31).

Anant Raj, Abhishek Kumar, Youssef Mroueh, Tom Fletcher, and Bernhard Schoelkopf (2017). 'Local Group Invariant Representations via Orbit Embeddings'. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (pages 72, 77).

Carl Edward Rasmussen and Zoubin Ghahramani (2001). 'Occam's Razor'. In: *Advances in Neural Information Processing Systems 13* (pages 12, 13).

Carl Edward Rasmussen and Hannes Nickisch (2010). 'Gaussian Processes for Machine Learning (GPML) Toolbox'. *Journal of Machine Learning Research* (page 21).

Carl Rasmussen and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press (pages 10, 13, 15, 16, 18–20, 39–41, 65).

Sachin Ravi and Hugo Larochelle (2017). 'Optimization as a model for few-shot learning'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (pages 133, 135, 140, 156, 160, 161, 173, 178, 182, 184, 236).

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi (2017). 'Learning multiple visual domains with residual adapters'. In: *Advances in Neural Information Processing Systems* (page 189).

Mengye Ren, Sachin Ravi, Eleni Triantafillou, Jake Snell, Kevin Swersky, Josh B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel (2018). 'Meta-Learning for Semi-Supervised Few-Shot Classification'. In: *International Conference on Learning Representations* (page 199).

Danilo Jimenez Rezende and Shakir Mohamed (2015). 'Variational Inference with Normalizing Flows'. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning* (page 30).

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra (2014). 'Stochastic Backpropagation and Approximate Inference in Deep Generative Models'. In: *Proceedings of the 31st International Conference on Machine Learning* (pages 15, 23, 25, 28, 97, 167).

Geoffrey Roeder, Yuhuai Wu, and David K Duvenaud (2017). 'Sticking the Landing: Simple, Lower-Variance Gradient Estimators for Variational Inference'. In: *Advances in Neural Information Processing Systems 30* (page 28).

Michal Rolinek, Dominik Zietlow, and Georg Martius (2019). 'Variational Autoencoders Recover PCA Directions (by Accident)'. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (page 25).

Olaf Ronneberger, Philipp Fischer, and Thomas Brox (2015). 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (page 92).

Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed (2018). 'Distribution Matching in Variational Inference'. *arXiv* (pages 97–99, 129, 201).

Sam Roweis (1998). 'EM Algorithms for PCA and SPCA'. In: *Advances in Neural Information Processing Systems 10* (page 23).

Donald B. Rubin and Dorothy T. Thayer (1982). 'EM algorithms for ML factor analysis'. *Psychometrika* (page 23).

Sebastian Ruder (2017). 'An Overview of Multi-Task Learning in Deep Neural Networks'. *CoRR* (page 135).

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei (2015).

'ImageNet Large Scale Visual Recognition Challenge'. *International Journal of Computer Vision (IJCV)* (pages 136, 137, 156).

Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell (2019). 'Meta-Learning with Latent Embedding Optimization'. In: *International Conference on Learning Representations* (pages 25, 182).

Yunus Saatçi (2011). 'Scalable Inference for Structured Gaussian Process Models'. PhD thesis. University of Cambridge (page 49).

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma (2017). 'PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications'. In: *Proceedings of the 5th International Conference on Learning Representations* (pages 23, 29).

Tim Salimans, Diederik P. Kingma, and Max Welling (2015). 'Markov Chain Monte Carlo and Variational Inference: Bridging the gap'. In: *Proceedings of the 32nd International Conference on Machine Learning* (page 30).

John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck (2016). 'Probabilistic programming in Python using PyMC3'. *PeerJ Computer Science* (page 154).

Yves-Laurent Kom Samo and Stephen Roberts (2015). 'Generalized Spectral Kernels'. *arXiv preprint arXiv:1506.02236* (page 48).

Victor Garcia Satorras and Joan Bruna Estrach (2018). 'Few-Shot Learning with Graph Neural Networks'. In: *International Conference on Learning Representations* (page 182).

Jürgen Schmidhuber (1987). 'Evolutionary principles in self-referential learning'. PhD thesis. Technische Universität München (pages 134, 167).

Bernhard Schölkopf, Chris Burges, and Vladimir Vapnik (1996). 'Incorporating invariances in support vector learning machines'. In: *Artificial Neural Networks — ICANN 96* (page 71).

Bernhard Schölkopf, Patrice Simard, Alexander J. Smola, and Vladimir Vapnik (1998). 'Prior knowledge in support vector kernels'. In: *Advances in Neural Information Processing Systems 10* (page 72).

Bernhard Schölkopf and Alexander J. Smola (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press (page 18).

Matthias Seeger (2003). 'Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations'. PhD thesis. University of Edinburgh (page 13).

Matthias Seeger, Christopher K. I. Williams, and Neil D. Lawrence (2003). 'Fast Forward Selection to Speed Up Sparse Gaussian Process Regression'. In: *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics* (pages 37, 40, 41, 53, 200).

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis (2018). 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play'. *Science* (page 196).

Patrice Y. Simard, Yann A. Le Cun, John S. Denker, and Bernard Victorri (2000). 'Transformation invariance in pattern recognition: Tangent distance and propagation'. *International Journal of Imaging Systems and Technology* (pages 71, 87, 88).

Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker (1992). 'Tangent Prop – A formalism for specifying selected invariances in an adaptive network'. In: *Advances in Neural Information Processing Systems 4* (page 72).

K. Simonyan and A. Zisserman (2015). 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: *International Conference on Learning Representations* (pages 136, 160, 232).

Alexander J. Smola and Peter Bartlett (2001). 'Sparse greedy Gaussian process regression'. In: *Advances in Neural Information Processing Systems 13*. Citeseer (page 37).

Alexander J. Smola and Bernhard Schölkopf (2000). 'Sparse greedy matrix approximation for machine learning'. In: Max-Planck-Gesellschaft (page 46).

Jake Snell, Kevin Swersky, and Richard Zemel (2017). 'Prototypical networks for few-shot learning'. In: *Advances in Neural Information Processing Systems* (pages 139, 160, 161, 179, 184).

Edward Snelson (2007). 'Flexible and efficient Gaussian process models for machine learninglson'. PhD thesis. University College London (pages 38, 63).

Edward Snelson and Zoubin Ghahramani (2006). 'Sparse Gaussian Processes using Pseudo-inputs'. In: *Neural Information Processing Systems* (pages 16, 37, 41, 42, 51, 53, 54, 200).

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther (2016). 'Ladder Variational Autoencoders'. In: *Advances in Neural Information Processing Systems 29* (page 117).

C. Spearman (1904). '"General Intelligence," Objectively Determined and Measured'. *The American Journal of Psychology* (pages 23, 24).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. *Journal of Machine Learning Research* (page 232).

Nitish Srivastava and Ruslan R Salakhutdinov (2013). 'Discriminative transfer learning with tree-based priors'. In: *Advances in Neural Information Processing Systems* (pages 141, 143, 152, 159).

Michael L. Stein (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Springer (page 48).

Christian Steinruecken, Emma Smith, David Janz, James Lloyd, and Zoubin Ghahramani (2019). 'The Automatic Statistician'. In: *Automated Machine Learning* (page 19).

Ingo Steinwart and Andreas Christmann (2008). *Support vector machines*. Springer Science & Business Media (page 18).

Amos J. Storkey (1999). 'Truncated covariance matrices and Toeplitz methods in Gaussian processes'. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)* (page 49).

Freek Stulp and Olivier Sigaud (2015). 'Many regression algorithms, one unified model: A review'. *Neural Networks* (page 18).

Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori (2012). 'Density-ratio matching under the Bregman divergence: a unified framework of density-ratio estimation'. *Annals of the Institute of Statistical Mathematics* (pages 129, 176, 233).

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse (2019). 'Functional Variational Bayesian Neural Networks'. In: *International Conference on Learning Representations* (page 195).

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales (2018). 'Learning to Compare: Relation Network for Few-Shot Learning'. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pages 139, 184).

Esteban G. Tabak and Cristina V. Turner (2013). 'A Family of Nonparametric Density Estimation Algorithms'. *Communications on Pure and Applied Mathematics* (page 30).

Esteban G. Tabak and Eric Vanden-Eijnden (2010). 'Density estimation by dual ascent of the log-likelihood'. *Commun. Math. Sci.* (Page 30).

The GPy authors (2012). *GPy: A Gaussian process framework in python*. `http://github.com/SheffieldML/GPy` (page 21).

Sebastian Thrun (1996). 'Is Learning The n-th Thing Any Easier Than Learning The First?' In: *Advances in Neural Information Processing Systems 8* (page 134).

Sebastian Thrun and Lorien Pratt (2012). *Learning to learn*. Springer Science & Business Media (pages 134, 167).

Michael E. Tipping and Christopher Bishop (1999). 'Probabilistic Principal Component Analysis'. *Journal of the Royal Statistical Society, Series B* (page 23).

Michalis K. Titsias (2009a). 'Variational Learning of Inducing Variables in Sparse Gaussian Processes'. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (pages 14, 23, 37, 43, 46, 56, 63, 79, 200).

— (2009b). *Variational Model Selection for Sparse Gaussian Process Regression*. Technical report. University of Manchester (pages 43, 44, 53).

Michalis K. Titsias and Neil D. Lawrence (2010). 'Bayesian Gaussian Process Latent Variable Model'. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (page 23).

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf (2018). 'Wasserstein Auto-Encoders'. In: *International Conference on Learning Representations* (page 33).

Jakub Tomczak and Max Welling (2018). 'VAE with a VampPrior'. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics* (pages 98–100, 113, 115, 117, 118, 128, 195, 228).

Alessandra Tosi, Søren Hauberg, Alfredo Vellido, and Neil D. Lawrence (2014). 'Metrics for Probabilistic Geometries'. In: *Uncertainty in Artificial Intelligence* (page 24).

Dustin Tran, Rajesh Ranganath, and David M. Blei (2016). 'The Variational Gaussian Process'. In: *Proceedings of the 4th International Conference on Learning Representations* (page 117).

Eleni Triantafillou, Richard Zemel, and Raquel Urtasun (2017). 'Few-Shot Learning Through an Information Retrieval Lens'. In: *Advances in Neural Information Processing Systems 30* (page 184).

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle (2018). 'Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples'. In: *Workshop on Meta-Learning (MetaLearn 2018) at the 32nd Conference on Neural Information Processing Systems* (page 199).

Brian Trippe and Richard Turner (2018). 'Overpruning in Variational Bayesian Neural Networks'. *arXiv preprint arXiv:1801.06230* (page 183).

George Tucker, Andriy Mnih, Chris J. Maddison, John Lawson, and Jascha Sohl-Dickstein (2017). 'REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models'. In: *Advances in Neural Information Processing Systems 30* (page 30).

Richard E. Turner and Maneesh Sahani (2011). 'Two problems with variational expectation maximisation for time-series models'. In: *Bayesian Time Series Models*. Chapter 5 (pages 33, 53, 76, 183, 194, 195, 200).

Arash Vahdat, William Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash (2018). 'DVAE++: Discrete Variational Autoencoders with Overlapping Transformations'. In: *Proceedings of the 35th International Conference on Machine Learning* (page 195).

Mark van der Wilk, Matthias Bauer, ST John, and James Hensman (2018). 'Learning Invariances using the Marginal Likelihood'. In: *Advances in Neural Information Processing Systems 31* (pages 3, 5, 69).

Mark van der Wilk, Carl Edward Rasmussen, and James Hensman (2017). 'Convolutional Gaussian Processes'. In: *Advances in Neural Information Processing Systems* (pages 40, 67, 69, 72, 80).

Joaquin Vanschoren (2018). 'Meta-Learning: A Survey'. *CoRR* (page 134).

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol (2008). 'Extracting and Composing Robust Features with Denoising Autoencoders'. In: *Proceedings of the 25th International Conference on Machine Learning* (page 28).

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra et al. (2016). 'Matching networks for one shot learning'. In: *Advances in Neural Information Processing Systems* (pages 133, 139, 156, 160, 173, 182, 184, 188, 236, 237).

John von Neumann (1951). 'Various techniques used in connection with random digits'. In: *Monte Carlo Method* (pages 102, 104, 111).

Ekaterina Vylomova, Laura Rimell, Trevor Cohn, and Timothy Baldwin (2016). 'Take and Took, Gaggle and Goose, Book and Read: Evaluating the Utility of Vector Differences for Lexical Relation Learning'. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (page 24).

Grace Wahba (1990). 'Spline Models for Observational Data' (page 37).

Martin J. Wainwright and Michael I. Jordan (2007). 'Graphical Models, Exponential Families, and Variational Inference'. *Foundations and Trends in Machine Learning* (page 14).

Yu-Xiong Wang, Ross Girshick, Martial Herbert, and Bharath Hariharan (2018). 'Low-shot Learning from Imaginary Data'. In: *Computer Vision and Pattern Recognition (CVPR)* (page 139).

Yaqing Wang and Quanming Yao (2019). 'Few-shot Learning: A Survey'. *CoRR* (page 138).

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli (2004). 'Image quality assessment: from error visibility to structural similarity'. *IEEE transactions on image processing* (page 186).

Florian Wenzel, Theo Galy-Fajou, Christan Donner, Marius Kloft, and Manfred Opper (2018). 'Efficient Gaussian Process Classification Using Pòlya-Gamma Data Augmentation'. *arXiv preprint arXiv:1802.06383* (page 85).

Christopher K. I. Williams (1998). 'Computation with Infinite Neural Networks'. *Neural Computation* (pages 10, 195).

Christopher K. I. Williams and Matthias Seeger (2001). 'Using the Nyström method to speed up kernel machines'. In: *Advances in Neural Information Processing Systems 13 (NIPS 2000)* (page 40).

Ronald J. Williams (1992). 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. *Machine Learning* (pages 29, 126).

Andrew G. Wilson, Elad Gilboa, John P Cunningham, and Arye Nehorai (2014). 'Fast kernel learning for multidimensional pattern extrapolation'. In: *Advances in Neural Information Processing Systems 27* (page 49).

Andrew G. Wilson, Zhiting Hu, Ruslan R. Salakhutdinov, and Eric P. Xing (2016a). 'Stochastic Variational Deep Kernel Learning'. In: *Advances in Neural Information Processing Systems 29* (page 49).

Andrew G. Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing (2016b). 'Deep Kernel Learning'. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (pages 19, 49).

Andrew G. Wilson and Hannes Nickisch (2015). 'Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)'. In: *Proceedings of the 32nd International Conference on Machine Learning* (page 49).

Han Xiao, Kashif Rasul, and Roland Vollgraf (28th2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms* (page 115).

Zichao Yang, Alexander J. Smola, Le Song, and Andrew G. Wilson (2015). 'A la Carte - Learning Fast Kernels'. In: *Artificial Intelligence and Statisticsa* (page 49).

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss (2005). 'Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms'. *IEEE Transactions on Information Theory* (page 32).

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn (2018). 'Bayesian Model-Agnostic Meta-Learning' (pages 140, 184).

Yunong Zhang, W. E. Leithead, and D. J. Leith (2005). 'Time-series Gaussian Process Regression Based on Toeplitz Computation of O(N2) Operations and O(N)-level Storage'. In: *Proceedings of the 44th IEEE Conference on Decision and Control* (page 49).

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola (2017). 'Deep sets'. In: *Advances in Neural Information Processing Systems* (page 175).

Shengjia Zhao, Jiaming Song, and Stefano Ermon (2019). 'InfoVAE: Balancing Learning and Inference in Variational Autoencoders'. In: (pages 33, 195).

APPENDIX

A

RESAMPLED PRIORS FOR VARIATIONAL AUTOENCODERS

A.1 EXPERIMENTAL DETAILS

A.1.1 *Network architectures*

In general, we decided to use standard architectures. We observed that more complicated networks can overfit quite drastically, especially on staticMNIST, which is not dynamically binarised.

Notation.

For all networks, we specify the input size and then consecutively the output sizes of the individual layers separated by a " $-$ ". Potentially, outputs are reshaped to convert between convolutional layers (abbreviated by "CNN") and fully connected layers (abbreviated by "MLP"). When we nest networks, e.g. by writing $p(\mathbf{x} \mid \mathbf{z}_1, \mathbf{z}_2) = \mathrm{MLP}[[\mathrm{MLP}[d_{\mathbf{z}} - 300], \mathrm{MLP}[d_{\mathbf{z}} - 300]] - 300 - 28 \times 28]$, we mean that first the two inputs/conditioning variables, $\mathbf{z}_1$ and $\mathbf{z}_2$ in this case, are transformed by neural networks, here an $\mathrm{MLP}[d_{\mathbf{z}} - 300]$, and their concatenated outputs (indicated by $[\cdot, \cdot]$) are subsequently used as an input to another network, in this case with hidden layer of 300 units and reshaped output 28×28.

VAE ($L = 1$).

For the single latent layer VAE, we used an MLP with two hidden layers of 300 units each for both the encoder and the decoder. The latent space was chosen to be $d_{\mathbf{z}} = 50$ dimensional and the nonlinearity for the hidden layers was `tanh`; the likelihood was chosen to be Bernoulli. The encoder parameterises the mean μ and the log standard deviation $\log \sigma$ of the diagonal Normal variational posterior.

$$q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_{\mathbf{z}}(\mathbf{x}), \sigma_{\mathbf{z}}(\mathbf{x}))$$
$$p(\mathbf{x} \mid \mathbf{z}) = \mathrm{Bernoulli}(\mathbf{x}; \mu_{\mathbf{x}}(\mathbf{z}))$$

which are given by:

$$\begin{aligned}
\mu_{\mathbf{z}}(\mathbf{x}) &= \text{MLP}[28 \times 28 - 300 - 300 - d_{\mathbf{z}}] \\
\log \sigma_{\mathbf{z}}(\mathbf{x}) &= \text{MLP}[28 \times 28 - 300 - 300 - d_{\mathbf{z}}] \\
\mu_{\mathbf{x}}(\mathbf{z}) &= \text{MLP}[d_{\mathbf{z}} - 300 - 300 - 28 \times 28]
\end{aligned}$$

where the networks for $\mu_z(\mathbf{x})$ and $\log \sigma_z(\mathbf{x})$ are shared up to the last layer. In the following, we will abbreviate this as follows:

$$\begin{aligned}
q(\mathbf{z} \mid \mathbf{x}) &= \text{MLP}[28 \times 28 - 300 - 300 - d_{\mathbf{z}}] \\
p(\mathbf{x} \mid \mathbf{z}) &= \text{MLP}[d_{\mathbf{z}} - 300 - 300 - 28 \times 28]
\end{aligned}$$

HVAE ($L = 2$).

For the hierarchical VAE with two latent layers, we used two different architectures, one based on MLPs and the other on convolutional layers. Both were inspired by the architectural choices of Tomczak and Welling (2018), however, we chose to use simpler models than them with fewer layers and regular nonlinearities instead of gated units to avoid overfitting.

The MLP was structured very similarly to the single layer case:

$$\begin{aligned}
q(\mathbf{z}_2 \mid \mathbf{x}) &= \text{MLP}[28 \times 28 - 300 - 300 - d_{\mathbf{z}}] \\
q(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{z}_2) &= \text{MLP}[[\text{MLP}[28 \times 28 - 300], \text{MLP}[d_{\mathbf{z}} - 300]] - 300 - d_{\mathbf{z}}] \\
p(\mathbf{z}_1 \mid \mathbf{z}_2) &= \text{MLP}[d_{\mathbf{z}} - 300 - 300 - d_{\mathbf{z}}] \\
p(\mathbf{x} \mid \mathbf{z}_1, \mathbf{z}_2) &= \text{MLP}[[\text{MLP}[d_{\mathbf{z}} - 300], \text{MLP}[d_{\mathbf{z}} - 300]] - 300 - 28 \times 28]
\end{aligned}$$

We again use `tanh` nonlinearities in hidden layers of the MLP and a Bernoulli likelihood model.

ConvHVAE ($L = 2$).

The overall model structure is similar to the HVAE but instead of MLPs we use CNNs with strided convolutions if necessary. To avoid imbalanced up/down-sampling we chose a kernel size of 4 which works well with strided up/down-

convolutions. As nonlinearities we used `ReLU` activations after convolutional layers and `tanh` nonlinearities after MLP layers.

$$
\begin{aligned}
q(\mathbf{z}_2 \mid \mathbf{x}) &= \text{MLP}\left[\text{CNN}\left[28 \times 28 \times 1 - 14 \times 14 \times 32 - 7 \times 7 \times 32 - \right.\right. \\
&\qquad \left.\left. 4 \times 4 \times 32\right] - d_{\mathbf{z}}\right] \\
q(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{z}_2) &= \text{MLP}\left[\left[\text{CNN}\left[28 \times 28 \times 1 - 14 \times 14 \times 32 - 7 \times 7 \times 32 - \right.\right.\right. \\
&\qquad \left.\left.\left. 4 \times 4 \times 32\right], \text{MLP}[d_{\mathbf{z}} - 4 \times 4 \times 32]\right] - 300 - d_{\mathbf{z}}\right] \\
p(\mathbf{z}_1 \mid \mathbf{z}_2) &= \text{MLP}[d_{\mathbf{z}} - 300 - 300 - d_{\mathbf{z}}] \\
p(\mathbf{x} \mid \mathbf{z}_1, \mathbf{z}_2) &= \text{CNN}\left[\text{MLP}[[\text{MLP}[d_{\mathbf{z}} - 300], \text{MLP}[d_{\mathbf{z}} - 300]] - 4 \times 4 \times 32] - \right. \\
&\qquad \left. 7 \times 7 \times 32 - 14 \times 14 \times 32 - 32 \times 32 \times 1\right]
\end{aligned}
$$

Acceptance function $a(\mathbf{z})$.

For $a(\mathbf{z})$ we use a very simple MLP architecture,

$$a(\mathbf{z}) = \text{MLP}[d_{\mathbf{z}} - 100 - 100 - 1],$$

again with `tanh` nonlinearities in the hidden layers and a `logistic` nonlinearity on the output to ensure that the final value is in the range $[0, 1]$.

RealNVP.

For the RealNVP prior/proposal we employed the reference implementation from `TensorFlow Probablity` (Dillon et al., 2017). We used a stack of 4 RealNVPs with two hidden MLP layers of 100 units each and performed reordering permutations in-between individual RealNVPs, as the RealNVP only transforms half of the variables.

Masked autoregressive flows.

For the Masked Autoregressive Flows (Papamakarios et al., 2017) (MAF) prior/proposal we use a stack of 5 MAFs with $\text{MLP}[100 - 100]$ each and again use the reference implementation from `TensorFlow Probablity` (Dillon et al., 2017). Random permutations are employed between individual MAF blocks (Papamakarios et al., 2017).

B

DISCRIMINATIVE FEW-SHOT LEARNING USING PROBABILISTIC MODELS

B.1 EXPERIMENTAL DETAILS

B.1.1 *Network architecture and training: ResNet inspired*

The network architecture is inspired by the ResNet-34 architecture for ImageNet (He et al., 2016) that uses convolution blocks, with two convolutions each, that are bridged by skip connections. As a base, we utilise the example code[1] provided by `tensorpack` (https://github.com/ppwwyyxx/tensorpack), a neural network training library built on top of `tensorflow` (Martín Abadi et al., 2015). We adapt the number of features as well as the size of the last fully connected layer to account for the smaller number of training samples and training classes. The final architecture is detailed in Table B.1.

ResNet-34 inspired network for *mini*ImageNet

Output size	Layers
$84 \times 84 \times 3$	Input patch
$42 \times 42 \times 32$	$5 \times 5, 32$, stride 2
$42 \times 42 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$
$21 \times 21 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 4$
$11 \times 11 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 6$
$6 \times 6 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 3$
256	global average pooling
$\widetilde{C}$	fully connected, softmax

Table B.1: Network architecture for a ResNet-34 inspired network for few-shot learning on *mini*ImageNet. All unnamed layers are 2D convolutions with stated kernel size and padding SAME; the output of the shaded layer corresponds to $\Phi_\varphi(\mathbf{x})$, the feature space representation of the image $\mathbf{u}$, which is used as input for probabilistic few-shot learning.

The network is trained using a decaying learning rate schedule and momentum SGD and is implemented in `tensorpack` using `tensorflow`.

1 https://github.com/ppwwyyxx/tensorpack/tree/master/examples/ResNet

B.1.2 *Network architecture and training: VGG inspired*

VGG-style Network for CIFAR-100

Output size	Layers
$32 \times 32 \times 3$	Input patch
$16 \times 16 \times 64$	$2\times$ (Conv2D, ELU), Pool
$8 \times 8 \times 64$	$2\times$ (Conv2D, ELU), Pool
$4 \times 4 \times 128$	$2\times$ (Conv2D, ELU), Pool
$2 \times 2 \times 128$	$2\times$ (Conv2D, ELU), Pool
$2 \times 2 \times 128$	Dropout (0.5)
256	FullyConnected, ELU
256	Dropout (0.5)
128	FullyConnected, ELU
$\widetilde{C}$	FullyConnected, SoftMax

VGG-style Network for *mini*ImageNet

Output size	Layers
$84 \times 84 \times 3$	Input patch
$42 \times 42 \times 32$	$2\times$ (Conv2D, ELU), Pool
$21 \times 21 \times 64$	$2\times$ (Conv2D, ELU), Pool
$11 \times 11 \times 128$	$2\times$ (Conv2D, ELU), Pool
$6 \times 6 \times 128$	$2\times$ (Conv2D, ELU), Pool
$3 \times 3 \times 128$	$2\times$ (Conv2D, ELU), Pool
$3 \times 3 \times 128$	Dropout (0.5)
512	FullyConnected, ELU
512	Dropout (0.5)
256	FullyConnected, ELU
$\widetilde{C}$	FullyConnected, SoftMax

Table B.2: Network architecture for VGG inspired networks for few-shot learning on *mini*ImageNet and CIFAR-100. All 2D convolutions have kernel size 3×3 and padding SAME; max-pooling is performed with stride 2. The output of the shaded layer corresponds to $\Phi_\varphi(\mathbf{x})$, the feature space representation of the image $\mathbf{x}$, which is used as input for probabilistic few-shot learning

The network architecture was inspired by the VGG networks (Simonyan and Zisserman, 2015), but does not employ batch normalisation (Ioffe and Szegedy, 2015). To speed up training, we employ exponential linear units (ELUs), which have been reported to lead to faster convergence as compared to ordinary ReLUs (Clevert et al., 2016). To regularise the networks, we employ dropout (Srivastava et al., 2014) and regularisation of the weights in the fully connected layers. The networks are trained with the ADAM optimiser (Kingma and Ba, 2015) with decaying learning rate.

The network is implemented in `tensorpack` using `tensorflow`.

C

META-LEARNING PROBABILISTIC INFERENCE FOR PREDICTION

C.1 JUSTIFICATION FOR THE CONTEXT-INDEPENDENT APPROXIMATION

In this section we lay out both theoretical and empirical justifications for the context-independent approximation detailed in Sections 7.2 and 8.2.

C.1.1 *Theoretical argument: density ratio estimation*

A principled justification for the approximation is best understood through the lens of density ratio estimation (Mohamed, 2018; Sugiyama et al., 2012). We denote the conditional density of each class as $p(\mathbf{x} \mid y = c)$ and assume equal a priori class probability $p(y = c) = 1/C$. Density ratio theory then uses Bayes' theorem to show that the optimal softmax classifier can be expressed in terms of the conditional densities (Mohamed, 2018; Sugiyama et al., 2012):

$$\text{Softmax}(y = c \mid \mathbf{x}) = \frac{\exp(\Phi_\varphi(\mathbf{x})^\top \mathbf{w}_c)}{\sum_{c'} \exp(\Phi_\varphi(\mathbf{x})^\top \mathbf{w}_{c'})} = p(y = c \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = c)}{\sum\limits_{c'} p(\mathbf{x} \mid y = c')}, \tag{C.1}$$

This implies that the optimal classifier will construct estimators for the conditional density for each class, that is $\exp(\Phi_\varphi(\mathbf{x})^\top \mathbf{w}_c) \propto p(\mathbf{x} \mid y = c)$. Importantly for our approximation, notice that these estimates are constructed *independently* for each class, similarly to training a naive Bayes classifier. VERSA mirrors this optimal form using:

$$\log p(\mathbf{x} \mid y = c) \propto \Phi_\varphi(\mathbf{x})^\top \mathbf{w}_c, \tag{C.2}$$

where $\mathbf{w}_c \sim q_v\left(\mathbf{w} \mid \{\mathbf{x}_n \mid y_n = c\}\right)$ for each class in a given task. Under ideal conditions (i.e., if one could perfectly estimate $p(\mathbf{x} \mid y = c)$), the context-independent assumption holds, further motivating our design.

c.1.2 *Empirical justification*

Here we detail a simple experiment to evaluate the validity of the context-independent inference assumption. The goal of the experiment is to examine if weights may be context-independent without imposing the assumption on the amortisation network. To see this, we randomly generate fifty tasks from a dataset, where classes may appear a number of times in different tasks. We then perform free-form (non-amortised) variational inference on the weights for each of the tasks, with a Gaussian variational distribution:

$$q_v\left(W^{(t)} \mid \mathcal{D}^{(t)}, \varphi\right) = \mathcal{N}\left(W^{(t)}; \mu_v^{(t)}, \sigma_v^{(t)2}\right). \tag{C.3}$$

If the assumption is reasonable, we may expect the distribution of the weights of a specific class to be similar regardless of the additional classes in the task.

We examine 5-way classification in the MNIST dataset. We randomly sample and fix fifty such tasks. We train the model twice using the same feature extraction network used in the few-shot classification experiments, and fix the d_φ to be 16 and 2. We then train the model in an episodic manner by mini-batching tasks at each iteration. The model is trained to convergence, and achieves 99% accuracy on held out test examples for the tasks. After training is complete we examine the optimized $\mu_v^{(t)}$ for each class in each task.

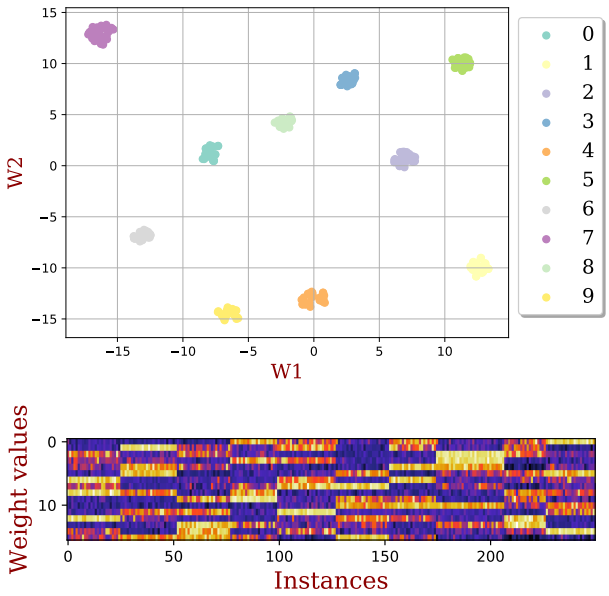


Figure C.1: Visualising the learned weights for $d_\varphi = 16$. *(top)* Weight dimensionality is reduced using T-SNE (Maaten and Hinton, 2008). Weights are colored according to class. *(bottom)* Each weight represents one column of the image. Weights are grouped by class.

Figure C.1 *(top)* shows a t-SNE (Maaten and Hinton, 2008) plot for the 16-dimensional weights. We see that when reduced to two dimensions, the weights cluster according to class. Figure C.1 *(bottom)* visualises the weights in their original space. In this plot, weights from the same class are grouped together, and clear similarity patterns are evident across the image, showing that weights from the same class have similar means across tasks.

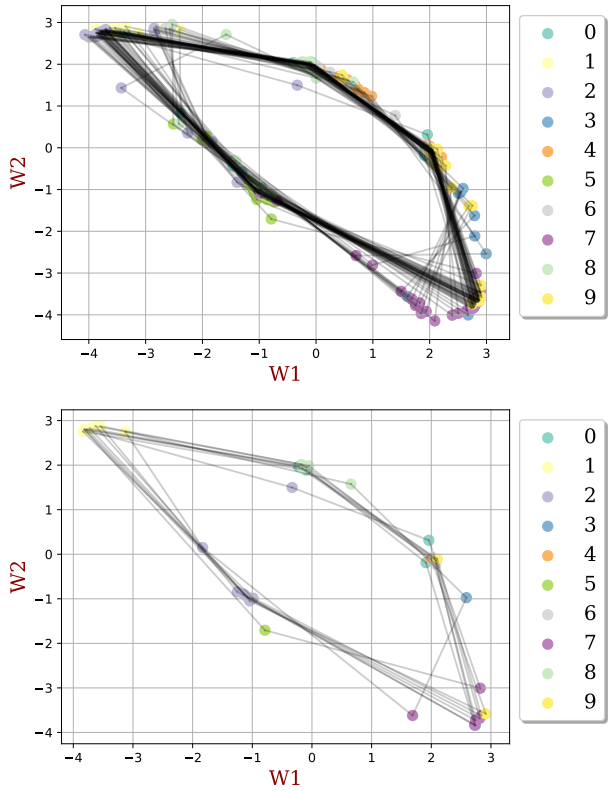


Figure C.2: Visualising the task weights for $d_\varphi = 2$. *(top)* All training tasks. *(bottom)* Only training tasks containing both '1's and '2's.

Figure C.2 details the task weights in 2-dimensional space. Here, each pentagon represents the weight means learned for one training task, where the nodes of the pentagon are colored according to the class the weights represent. In Figure C.2 *(top)* we see that overall, the classes cluster in 2-dimensional space as well. However, there is some overlap (e.g., classes '1' and '2'), and that for some tasks a class-weight may appear away from the cluster. Figure C.2 *(bottom)* shows the same plot, but only for tasks that contain both class '1' and '2'. Here we can see that for these tasks, class '2' weights are all located away from their cluster.

This implies that each class-weights are typically well-approximated as being independent of the task. However, if the model lacks capacity to properly assign each set of class weights to different regions of space, for tasks where classes from similar regions of space appear, the inference procedure will 'move' one of the class weights to an 'empty' region of the space.

C.2 EXPERIMENTAL DETAILS

In this section we provide comprehensive details on the few-shot classification experiments.

C.2.1 *Omniglot Few-shot Classification Training Procedure*

Omniglot (Lake et al., 2011) is a few-shot learning dataset consisting of 1623 handwritten characters (each with 20 instances) derived from 50 alphabets. We follow a pre-processing and training procedure akin to that defined in (Vinyals et al., 2016). First the images are resized to 28×28 pixels and then character classes are augmented with rotations of 90 degrees. The training, validation and test sets consist of a random split of 1100, 100, and 423 characters, respectively. When augmented this results in 4400 training, 400 validation, and 1292 test classes, each having 20 character instances. For C-way-k_c-shot classification, training proceeds in an episodic manner. Each training iteration consists of a batch of one or more tasks. For each task C classes are selected at random from the training set. During training, k_c character instances are used as training inputs and 15 character instances are used as test inputs. The validation set is used to monitor the progress of learning and to select the best model to test, but does not affect the training process. Final evaluation of the trained model is done on 600 randomly selected tasks from the test set. During evaluation, k_c character instances are used as training inputs and k_c character instances are used as test inputs. We use the Adam (Kingma and Ba, 2015) optimiser with a constant learning rate of 0.0001 with 16 tasks per batch to train all models. The 5-way-5-shot and 5-way-1-shot models are trained for 80,000 iterations while the 20-way-5-shot model is trained for 60,000 iterations, and the 20-way-1-shot model is trained for 100,000 iterations. In addition, we use a Gaussian form for q and set the number of ψ samples to $L = 10$.

C.2.2 mini*ImageNet Few-shot classification training procedure*

*mini*ImageNet (Vinyals et al., 2016) is a dataset of 60,000 color images that is sub-divided into 100 classes, each with 600 instances. The images have dimensions of 84×84 pixels. For our experiments, we use the 64 training, 16 validation, and 20 test class splits defined by (Ravi and Larochelle, 2017). Training proceeds in the same episodic manner as with Omniglot. We use the Adam (Kingma and Ba, 2015) optimiser and a Gaussian form for q and set the number of ψ samples to $L = 10$. For the 5-way-5-shot model, we train using 4 tasks per batch for 100,000 iterations and use a constant learning rate of 0.0001. For the 5-way-1-shot model,

we train with 8 tasks per batch for 50,000 iterations and use a constant learning rate of 0.00025.

c.2.3 *Few-shot classification network architectures*

Tables C.1 to C.4 detail the neural network architectures for the feature extractor Φ_φ, amortisation network q_υ, and linear classifier ψ, respectively. The feature extraction network is very similar to that used in (Vinyals et al., 2016). The output of the amortisation network yields mean-field Gaussian parameters for the weight distributions of the linear classifier ψ. When sampling from the weight distributions, we employ the local-reparameterisation trick (Kingma et al., 2015), that is we sample from the implied distribution over the logits rather than directly from the variational distribution. To reduce the number of learned parameters, we share the feature extraction network Φ_φ with the pre-processing phase of the amortisation network q_υ.

Omniglot Shared Feature Extraction Network Φ_φ: $\mathbf{x} \mapsto \Phi_\varphi(\mathbf{x})$

Output size	Layers
$28 \times 28 \times 1$	Input image
$14 \times 14 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$7 \times 7 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$4 \times 4 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
256	flatten

Table C.1: Feature extraction network used for Omniglot few-shot learning. Batch normalisation and dropout with a keep probability of 0.9 used throughout.

***mini*ImageNet Shared Feature Extraction Network Φ_φ: $\mathbf{x} \mapsto \Phi_\varphi(\mathbf{x})$**

Output size	Layers
$84 \times 84 \times 1$	Input image
$42 \times 42 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$21 \times 21 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$10 \times 10 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$5 \times 5 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
256	flatten

Table C.2: Feature extraction network used for *mini*ImageNet few-shot learning. Batch normalisation and dropout with a keep probability of 0.5 used throughout.

Amortisation Network q_υ: $\{\mathbf{x}_1^c, \ldots, \mathbf{x}_{k_c}^c\} \mapsto (\mu_{\mathbf{w}^{(c)}}, \sigma^2_{\mathbf{w}^{(c)}})$		
Phase	**Output size**	**Layers**
feature extraction	$k \times 256$	shared feature network (θ)
instance pooling	256	mean
ψ weight distribution	256	$2 \times$ fully connected, ELU + linear fully connected to $\mu_{w^{(c)}}, \sigma^2_{w^{(c)}}$

Table C.3: Amortisation network used for Omniglot and *mini*ImageNet few-shot learning.

Linear Classifier ψ_t: $\Phi_\varphi(\mathbf{x}^*) \mapsto p(y^* \mid \mathbf{x}^*, \varphi, \psi_t)$	
Output size	**Layers**
256	Input features
C	fully connected, softmax

Table C.4: Linear classifier used for Omniglot and *mini*ImageNet few-shot learning.

C.3 SHAPENET EXPERIMENTATION DETAILS

C.3.1 *View reconstruction training procedure and network architectures*

ShapeNetCore v2 (Chang et al., 2015) is an annotated database of 3D objects covering 55 common object categories with ∼51,300 unique objects. For our experiments, we use 12 of the largest object categories. Refer to Table C.1 for a complete list. We concatenate all instances from all 12 of the object categories together to obtain a dataset of 37,108 objects. This concatenated dataset is then randomly shuffled and we use 70% of the objects (25,975 in total) for training, 10% for validation (3,710 in total) , and 20% (7,423 in total) for testing. For each object, we generate $V = 36$ views of 128×128 pixels each spaced evenly every 10 degrees in azimuth around the object. We then convert the rendered images to grey scale and reduce their size to be 32×32 pixels. Again, we train our model in an episodic manner. Each training iteration consists a batch of one or more tasks. For each task an object is selected at random from the training set. We train on a single view selected at random from the $V = 36$ views associated with each object and use the remaining 35 views to evaluate the objective function. We then generate 36 views of the object with a modified version of our amortisation network which is shown diagrammatically in Figure 8.3. To evaluate the system, we generate views and compute quantitative metrics over the entire test set. Tables C.2 to C.4 describe the network architectures for the encoder, amortisation, and generator networks, respectively. To train, we use the Adam (Kingma and Ba, 2015) optimiser with a constant learning rate of 0.0001 with 24 tasks per batch for 500,000 training iterations. In addition, we set $d_\upsilon = 256$, $d_\psi = 256$ and number of ψ samples to 1.

Object Category	sysnet ID	Instances
airplane	02691156	4045
bench	02828884	1813
cabinet	02933112	1571
car	02958343	3533
phone	02992529	831
chair	03001627	6778
display	03211117	1093
lamp	03636649	2318
speaker	03691459	1597
sofa	04256520	3173
table	04379243	8436
boat	04530566	1939

Table C.1: List of ShapeNet categories used in the VERSA view reconstruction experiments.

ShapeNet encoder Network Φ_φ: $y \mapsto \Phi_\varphi$

Output size	Layers
$32 \times 32 \times 1$	Input image
$16 \times 16 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$8 \times 8 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$4 \times 4 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
d_Φ	fully connected, RELU

Table C.2: Encoder network used for ShapeNet few-shot view reconstruction. No dropout or batch normalisation is used.

ShapeNet Amortisation Network q_υ: $(\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_k^{(t)}, y_1^{(t)}, \ldots, y_k^{(t)}) \mapsto (\mu_\psi, \sigma_\psi^2)$

Phase	Output size	Layers
υ_{pre}	$k \times d_\upsilon$	encoder network (υ)
concatenate h and X	$k \times (d_\psi + d_X)$	concat(h, X)
υ_{mid}	$k \times d_\upsilon$	2×2 fully connected, ELU
instance pooling	$1 \times d_\upsilon$	average
υ_{post}	$1 \times d_\upsilon$	$2\times$ fully connected, ELU
ψ distribution	d_ψ	fully connected linear layers to μ_ψ, σ_ψ^2

Table C.3: Amortisation network used for ShapeNet few-shot view reconstruction.

ShapeNet Generator Network (φ): $\mathbf{x} \mapsto p(y \mid \mathbf{x}, \varphi, \psi^{(t)})$

Output size	**Layers**
$d_\psi + d_x$	concat(ψ, x)
512	fully connected, RELU
1024	fully connected, RELU
$2 \times 2 \times 256$	reshape
$4 \times 4 \times 128$	deconv2d (3×3, stride 2, SAME, RELU)
$8 \times 8 \times 64$	deconv2d (3×3, stride 2, SAME, RELU)
$16 \times 16 \times 32$	deconv2d (3×3, stride 2, SAME, RELU)
$32 \times 32 \times 1$	deconv2d (3×3, stride 2, SAME, sigmoid)

Table C.4: Generator network used for ShapeNet few-shot learning. No dropout or batch normalisation are used.